RASCAL; A RUDIMENTARY ADAPTIVE SYSTEM FOR COMPUTER-AIDED LEARNING

by

John Christopher Stewart

United States
Naval Postgraduate School

THESIS

RASCAL

A RUDIMENTARY ADAPTIVE SYSTEM

FOR

COMPUTER-AIDED LEARNING

by

John Christopher Stewart

December 1970

RASCAL

A Rudimentary Adaptive System

for

Computer-Aided Learning

by

John Christopher Stewart
Lieutenant, United States Navy
B.E.E., Rensselaer Polytechnic Institute, 1966

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 1970

ABSTRACT

The requirements of a Computer-Aided Learning System which would be a reasonable assistant to the teacher are discussed. These ideas are implemented in a system entitled RASCAL, a Rudimentary Adaptive System for Computer-Aided Learning. RASCAL replaces prepared frames used in previous systems with a description of questions to be asked and a tree of alternatives that might be helpful in assisting a student in answering a question. The actual questions are generated as a function of the system's interaction with the student, as is the selection of the branch to follow in aiding the student. The results obtained to date, while not extensive in their scope, indicate that a system such as RASCAL can be useful in the classroom.

TABLE OF CONTENTS

I. INTRODUCTION ---------- 5

II. BACKGROUND ---------- 8

A. DEVELOPMENT OF COMPUTER-ASSISTED INSTRUCTION -- 8

B. ARGUMENTS FOR COMPUTER-ASSISTED INSTRUCTION --- 10

C. ARGUMENTS AGAINST COMPUTER-ASSISTED INSTRUCTION ---------- 12

D. CLASSIFICATION OF CAI SYSTEMS ---------- 14

E. PROBLEMS WITH COMPUTER-ASSISTED INSTRUCTION --- 19

III. FACTORS IN DESIGN ---------- 22

A. IDENTIFICATION OF CRITICAL ELEMENTS ---------- 22

B. SYSTEM RESPONSIBILITIES ---------- 25

1. The Teacher's Role ---------- 25

2. The Computer's Role ---------- 28

3. The Programmer's Role ---------- 31

4. Interface between Teacher and Programmer ------- 31

5. Interface between Teacher and Computer -------- 31

IV. IMPLEMENTATION ---------- 34

A. STRUCTURE OF RASCAL ---------- 34

1. Description of a Problem Type ---------- 37

2. Description of Trees ---------- 41

3. Advancement of the Student ---------- 46

4. Problem Generation ---------- 51

5. Generating an Answer ---------- 62

6. Interpreting Trees ---------- 63

B. DEVELOPMENT ---------- 64

V. CONCLUDING REMARKS ---------- 66

APPENDIX A RASCAL - The Computer's Side ---------- 71

APPENDIX B RASCAL - The Teacher's Side ---------- 78

COMPUTER PROGRAM ---------- 90

BIBLIOGRAPHY ---------- 147

INITIAL DISTRIBUTION LIST ---------- 150

FORM DD 1473 ---------- 151

I. INTRODUCTION

The development of the computer in the middle of the twentieth century is having an effect on society comparable to that of the printing press five hundred years ago. In the field of education, the effect has been to reverse the trend of standardization started by the printing press. In the early 1900's educators had already begun to question the worth of mass education as compared to the older form of individualized or tutorial education practiced at Oxford and Cambridge. However, no economically viable way of individualizing was available. The computer offers an economical means to individualize instruction. Considerable research has been conducted in the past ten years, attempting to develop a computer-based system which would individualize the learning environment. This research has collectively been entitled Computer-Assisted Instruction (CAI) or Computer-Aided Learning (CAL), with the former term predominating.

While many systems have been devised, CAI, to a large degree, remains a laboratory phenomenon. It is felt that part of the reason behind this is that while systems, to date, have developed a high degree of interaction with the student, they have little capacity, if any, to interact with the teacher. RASCAL is an attempt to define a more viable system by identifying the proper roles of the teacher and the computer in the type of symbiotic relationship discussed by J. C. R. Licklider.[1]

[1] Licklider, J. C. R., "Man-Computer Symbiosis," IEE Transactions on Human Factors, HFE-1, p. 4-11, March 1960.

The key to this, it is felt, is the elimination of the prepared set of frames that is at the core of many CAI systems. This set of frames is inaccessible to the teacher although teachers may have been consulted in the preparation. To replace this set of frames, RASCAL uses a tree structure which is designed to approximate the steps a teacher would follow in assisting a student in the solution. It is intended that the branches of the tree be interactively constructed, either by the teacher or by the computer, from a broad set of alternatives given it by the teacher. This interactiveness is essential in any endeavor such as teaching, where the methods which may succeed in one case, fail in another and where there is no recognized body of information on how the endeavor is best accomplished. The ability to modify the trees does not imply the necessity to do so. Thus, the teacher has the choice of using trees previously created and stored or of creating his own to better fit what he thinks should be done.

To achieve the desired relationships, RASCAL assigns certain responsibilities to the computer and to the teacher. The assignment of responsibility is discussed in detail in Section III. Briefly, the computer has been assigned the responsibilities of: 1) generating problems for the student, 2) deciding when the student is ready to proceed to more difficult concepts, 3) presentation of instructional frames and review material, 4) file management and manipulation, and 5) the identification of problem areas to the teacher. The teacher has the responsibilities of: 1) identifying to the computer the student's present state of advancement and

capability, 2) specifying the type of problems to be presented to the student and the conditions which make a problem difficult or easy, and 3) the construction of branches to be followed in assisting the student to answer the problem.

RASCAL is written in Programming Language/One (PL/I). Its construction is highly modular to facilitate modifying and extending the system. Possible modifications and extensions are dealt with in Section V.

II. BACKGROUND

A. DEVELOPMENT OF COMPUTER-ASSISTED INSTRUCTION

Computer-Assisted Instruction has evolved from the concept of Programmed Instruction (PI) first developed by Sidney L. Pressey at Ohio State University in the 1920's. Unfortunately, Pressey's ideas did not catch on until the 1950's or sufficient data might have been available to make the transition of the concepts to computer-based systems easier. A real interest in PI did not develop until the 1950's when Dr. B. F. Skinner at Harvard University presented his findings on learning. Skinner expressed concern that the present system of education delayed the reinforcement of a response by so much that the learning process was seriously impaired.[2]

As Skinner's ideas preceded the commercial availability of computers, they were implemented in textual form. The Programmed Instruction texts of the 1950's demonstrated that, properly written, tested and administered, they are valuable instructional aids. However, the benefits are not without their problems. They are problems of supervision and change. In addition to allowing the student the freedom to study at his own rate, they also allow him the freedom not to study at all. Because the answers must be included in the text, the student is tempted merely to quickly

---

[2]Skinner, B. F., The Technology of Teaching, p. 14-22, Appleton-Century-Crofts, 1968.

glance through the text simply looking at those answers he does not know and continuing on. Thus, the student is deprived of the opportunity to think his way through the questions. Furthermore, he is defeating the intent of any branches that may be included to provide him with a better understanding of the material. Because the PI text is "hard copy," it cannot be easily changed. Thus, extensive testing must be performed before the text is released to print. This testing is most easily accomplished under controlled conditions. Since the data is not collected in the environment in which the text is used, there is some question as to its validity. Attempting to collect this data in the environment in which the text is used adds to the effort required by the student in using it, since he must keep a diary of impressions. This only encourages the improper use of the text.

The development of the computer offered a means of correcting these defects in Programmed Instruction. The data handling of the capacity of the computer allows it: 1) to continually gather data on the text it is presenting and to effectively evaluate the performance of the lesson, and 2) to keep records on the students to monitor their progress and insure that they are studying properly. The computer's erasable memory (or "Type Set") allows the text to be easily changed. Furthermore, it can be programmed so that the student does not see the answer and is forced to reason it out for himself.

Thus, the earliest CAI systems were simply PI texts implemented on a small computer dedicated to presenting the material to students.

These systems did not make use of the logical capabilities of the computer and were, therefore, somewhat restricted in their capacity to provide individualized instruction. In addition, they proved to be rather expensive and were limited by the size of the computer. To provide a more individualized presentation and lower costs, these systems have given way to highly sophisticated systems, such as Socrates, which are implemented on large computers capable of serving hundreds of students and handling tasks other than CAI presentation.[3] In fact, people are now beginning to talk in terms of Educational Utilities. As an example of things to come, the U. S. Office of Education has let contracts to two major organizations interested in educational time-sharing. The charge of these contracts is to design a CAI system having 100,000 terminals located in a radius of 100 miles.

B. ARGUMENTS FOR COMPUTER-ASSISTED INSTRUCTION

The main argument for CAI is based on the partly proven, partly conjectured advantages of individualized instruction and the ability of the computer to provide this individualized instruction economically. Other arguments advanced by proponents of computers are mentioned in the succeeding paragraphs.

The computer can collect and evaluate data about curriculum and learning. This will enable us to discover information which, even if

---

[3] Training Research Laboratory University of Illinois, Report 12, Socrates, a Computer-Based Instructional System in Theory and Research, by L. M. Stolurow, June 1966.

CAI proves unfeasible, will allow us to improve our educational methods. Further, it will provide data on whether our present system is suitable for CAI implementation.

Once sufficient data has been accumulated and learning strategies identified, the computer will be in the position to assess the student's abilities and present the learning strategy best suited to the student. In a mass-educational environment, the strategy selected must be the one which will help the greatest number of the students.

The computer can interact with the student to a surprising degree. While present state-of-the-art causes the machine to sometimes penalize students with the right idea but the wrong words and reward students with the wrong idea but the right words, further developments may make this interaction more dynamic than can be achieved in a mass education environment.

Recent studies indicate that the formalism of classrooms required for mass education may be detrimental to the learning process. Learning should be an interesting experience and the formality of the classroom too often makes the experience distasteful for young, inquisitive and mischievous minds. The computer offers unique capabilities for reducing formality in education. Computers can and have been programmed to teach in a game-like environment.

The computer's responses are geared to the student, and therefore, provide an environment in which outside distractions do not affect the learning process as much as in the classroom where the student may

miss an important point if his mind wanders. The computer cannot proceed until it has the student's attention.

The computer is gifted with infinite patience and is never bothered by routine drudgery. Thus, it is better suited than the conventional teachers for administering drill and practice sessions.

The computer can further relieve the teacher of classroom drudgery by administering and grading quizzes. It can keep a record of student progress and assist the teacher in her lesson planning.

C. ARGUMENTS AGAINST COMPUTER-ASSISTED INSTRUCTION

Arguments against CAI concern themselves with the cost of CAI, the effectiveness of CAI and the possible adverse effects that CAI may have on the student.

While it is true that CAI is expensive, (costs vary between $400/student/year and $50/student/year) this does not imply that such costs will continue. This same argument ran rampant in the early days of computers when the choice was between electronics using tubes or electronics using transistors. Transistors did in fact cost large amounts, but then they were as experimental as CAI systems today. The costs of CAI are dropping and will most likely continue to drop as the systems themselves become more sophisticated.

The argument that CAI is ineffective is a generalization on the early failures of CAI systems and presupposes that improvements cannot or will not be made. However, the potentials of CAI are so great that one should not be willing to scrap the entire idea because of a few earlier failures.

There is a fear that CAI will stifle the gifted student and lead to a mediocrity of all students. This argument is based on the early CAI systems which were merely automated PI programs. These programs, which were written for the average student with branching used to handle fast or slow students, made little use of the computer's logical capabilities. Present day systems are more imaginative in their approach and continued sophistication will further erode the grounds under this argument.

It is also feared that CAI may produce antisocial children who will regard the computer as infallible. It is based on recent results which indicate that group interaction in school may play a very important part in the learning process of early school children. This argument is being countered by the development of CAI systems which encourage student interaction.[4] Since children do not seem to be as impressed as their parents with the technology of today, there is some question as to whether they will come to regard the computer as infallible. However, should the problem arise, there is the possibility that it can be corrected by programming the computer to make appropriate errors.

Furthermore, it is feared that CAI will take away the teachers' jobs. However, CAI systems make no pretense of their need for teacher assistance. They simply cannot handle all the possible low probability occurrences. Therefore, CAI should not be looked upon as a replacement for

---

[4]Byron, G. L. "Student to Student Interaction in Computer Time-Sharing Systems," Computers and Automation, v. 18, p. 16-19, March 1969.

teachers, but as a supplement which, working together with the teacher, will provide for better teaching than is presently possible.

D. CLASSIFICATION OF CAI SYSTEMS

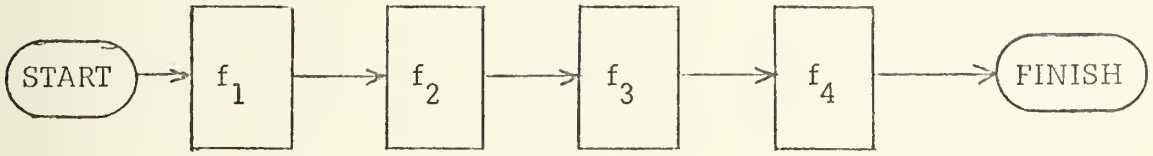
Computer-Assisted Instruction systems are classified into three major categories according to the intent of the system. The simplest type of system is the Drill and Practice. This type of system merely supplements the teacher. Questions are asked to the student, and if answered incorrectly, the correct answer is supplied. However, no attempt is made to identify or correct the error in understanding that caused the incorrect response to be given. It is the most superficial and accordingly the most developed and economical. However, the value of this type of system should not be underestimated. Many types of problems fall into a category which requires extensive practice to learn the basic algorithms with speed and accuracy. In these areas, Drill and Practice systems have been shown to be effective.

At the next higher level of interaction is the Tutorial system. This type of system is designed to take over, from the teacher, the main responsibility for instruction. It improves upon the Drill and Practice system in that remedial aid is presented when a problem is incorrectly answered. In addition, it attempts to provide the student with an understanding of the ideas for which the problems are providing practice. This type of system exists, but in general, is not sufficiently sophisticated to fulfill its objective entirely. However, it can relieve the teacher of the

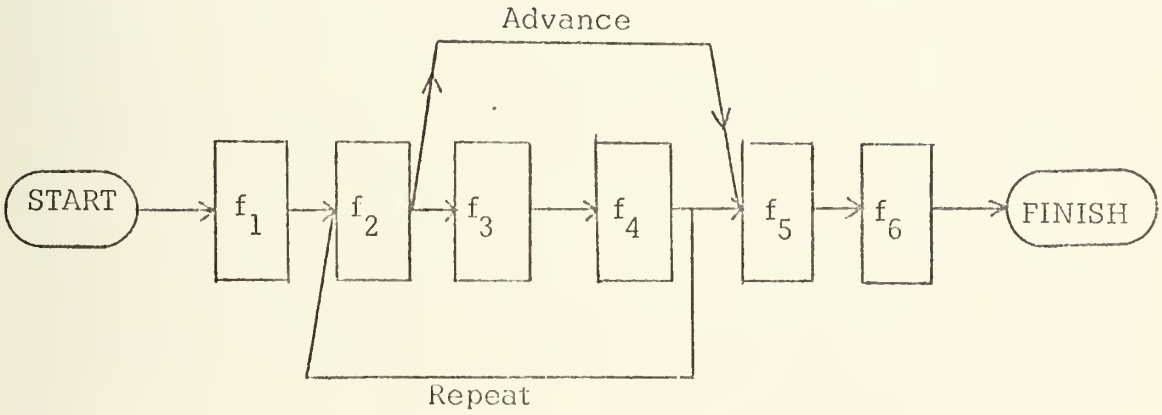
burden of teaching thirty students at once. This allows her more freedom to handle the individual cases where the computer fails.

The system with the highest level of machine responsibility is the Dialogue type. This type of system exists only as elementary prototypes, but provides the deepest level of interaction between the student and the computer. These types of programs represent the apex of Computer-Assisted Instruction systems. This type of system is not dependent upon the presentation of problems to determine where the student lacks understanding. It is designed to converse directly with the student, allowing the student to identify his problems. Ideally, the student would be able to ask any question on a given subject matter and the computer would be able to provide a satisfactory answer.

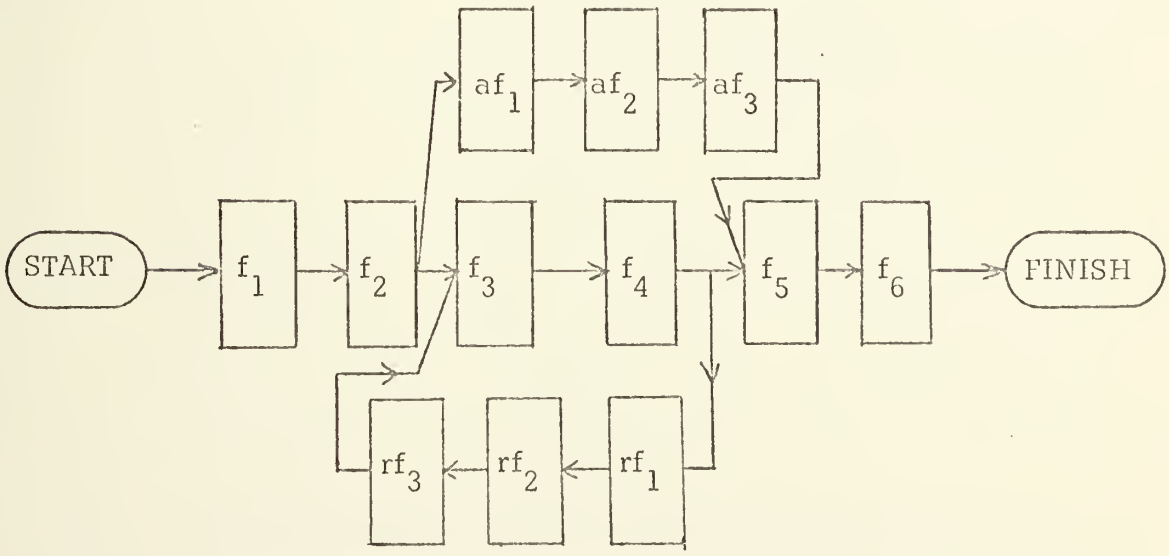
Computer-Assisted Instruction systems may also be classed according to their method of presenting material to the student. Figure 1 indicates the possible flow of a lesson in the most common methods of presentation. The earliest systems used a "linear" approach. The material was divided into a series of frames which were presented one at a time in a predescribed order. All frames were shown to each student. If the student failed to respond correctly to the material in a frame, he was simply given the correct response and the next frame was presented. A little more sophisticated are those systems which use a "simple branching" approach. Certain frames are designed to test the student's understanding of the subject being presented. If the student's answer indicates he has sufficiently grasped the idea, he is allowed to skip



a) Linear presentation

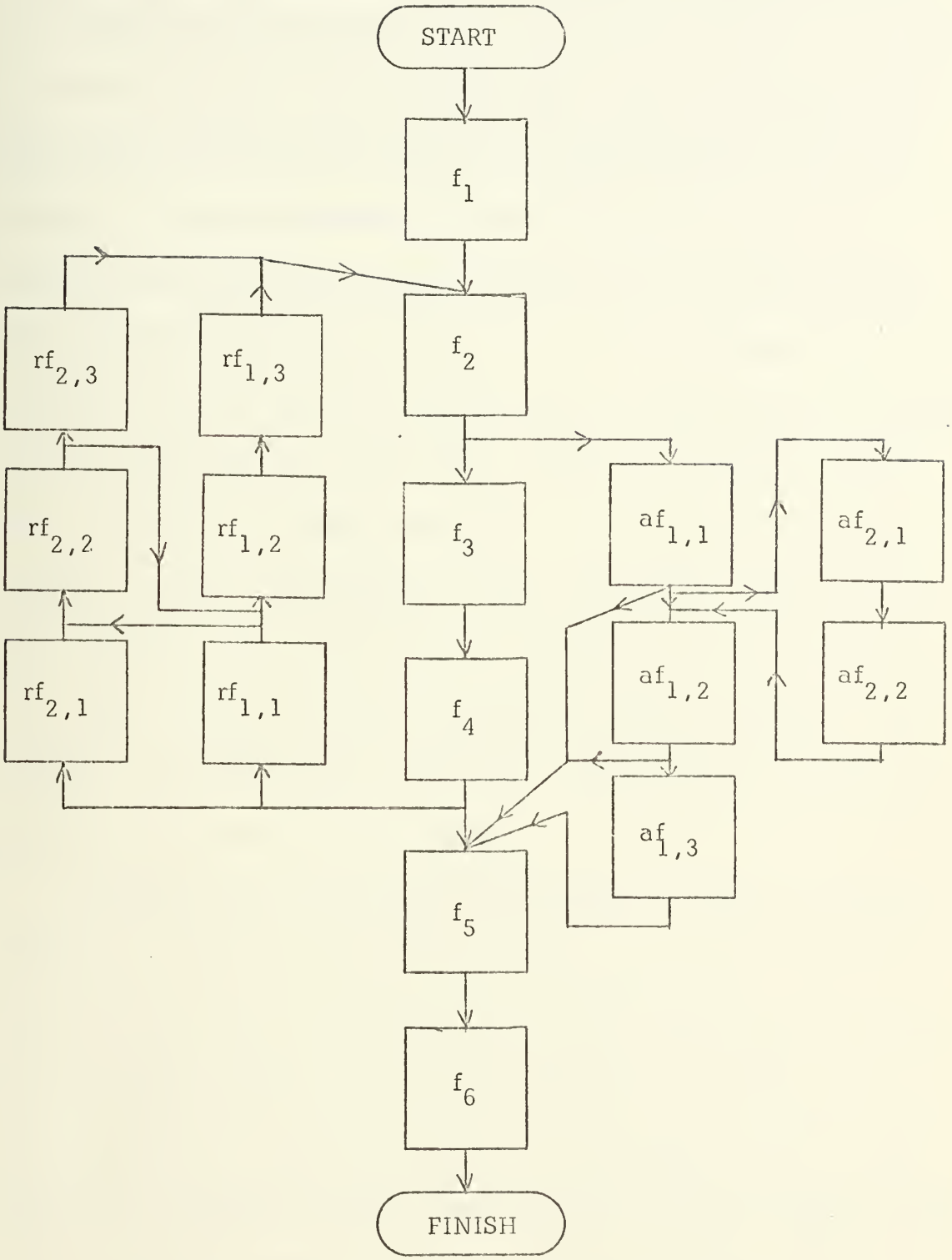


b) Simple Branching presentation



c) Complex Branching presentation

FIGURE 1



d) Multi-level Complex Branching presentation

FIGURE 1

ahead to the frame beginning the presentation of the next idea instead of continuing to the succeeding frame in the sequence which is designed to provide more practice with the same idea. Or if he has missed the idea completely, he may be returned to a previous frame to repeat the material. "Complex branching" systems not only allow the student to skip over frames, but also provide branches which allow the smarter or more interested student to go deeper into a subject, and branches which present remedial material to the student who is having trouble. The most complex type of system, based on this prepared frame idea, might be called "multi-level complex branching." This system is a "complex branching" system with a choice of paths at each node. Thus, if a student is having trouble and the first remedial presentation does not help, the machine can make a second attempt using different material. Also branches are provided within the branches which allow for further choice as to how the material is to be presented.

A system recently proposed, which appears to offer considerable advantage over the prepared frame system described above, is one in which the frames are replaced by a format of the questions to be asked, and the computer generates the actual questions as a function of the individual student's responses to questions presented to him.[5] As an extension to this type of system, the computer could also select remedial aids and advanced material for presentation to the student, from a set of

---

[5] Uhr, L., 24th Conference Proceedings of the Association for Computing Machinery, "Teaching Machine Programs that Generate Problems as a Function of Interaction with Students," p. 125-134, 1969.

such items given it, based on past experience with students who have responded in a similar manner. As an intermediate step in the construction of such a system, the remedial aids and advanced material might be given to the computer with an indication of when they are applicable. The computer could then construct Trees from these descriptions, select branches in the Trees, based on the student's response, and present the required material to the student. The development of this intermediate type of system was the goal in the construction of RASCAL.

E. PROBLEMS WITH COMPUTER ASSISTED-INSTRUCTION

At the root of the arguments against CAI is the fact that CAI has not developed at the rate envisioned for it. This is a result of underestimating the amount of time it would take to solve problems which of themselves are major areas of endeavor in the computer field.

First and foremost among these is the "natural language" problem. While many "question-answering" systems exist for limited subsets of English, there still exists no way for computers to understand precisely what questions have been asked them on a broad basis.

A second problem, particularly important to the elementary student, is the problem of oral language recognition. It is an accepted fact that smaller children have more trouble grasping concepts explained in writing than those explained orally. Similarly, they have more trouble expressing themselves in writing. Thus, a system with oral capability in both directions would be of immense value.

There exists no recognized body of fundamental theory about learning and retention. This makes it difficult to design a curriculum of study or to determine the best choice of a branch in reply to a given response. Thus, curriculum planning and course programming are done on a pragmatic basis, using methods which have given an indication of success in the past, but are not assured of success in the future. This may, in part, account for the unimpressive results in some studies comparing CAI with conventional methods.

Another problem and one where several approaches have been taken to solve it, is the familiar problem of communication between two very different disciplines. The people who are developing the systems are computer specialists, while the people who have the knowledge of curriculum and learning are educators. The problem of the one conveying his ideas to the other arises. This is no small problem since they speak different languages. The earliest method of solution was to form teams of educators and programmers to work side by side. A more sophisticated approach has been to construct languages in which the educators may construct their own CAI programs without the aid of a programmer. Everyone has had his own ideas as to what capabilities these languages should embody. As a result, there are some thirty different types on the market, e.g., Coursewriter by IBM and PLANIT by CDC. In the absence of a natural language recognizer, these languages probably are still too restricted for the average educator to ever desire to use. Unless a proposed change makes it readily apparent to the user, its value in making

his job easier, he will have nothing to do with it. The languages available to date do not show this feature.

III. FACTORS IN DESIGN

The intent of any CAI system must be to provide an inexpensive and effective aid to the teacher. To accomplish this goal, it seems necessary to identify the functions performed by a teacher in his job and the knowledge he requires to perform these functions. Only then would one be in a position to surmise which elements of teaching the computer would be able to handle, and to construct a system in which the computer would handle these elements.

A. IDENTIFICATION OF CRITICAL ELEMENTS

To describe the elements of a teacher's job as supervisory and tutorial is a gross over-simplification of the problem, but it does serve to warn that the teacher will remain a part of the system at least in lower grades. Present day computers would be of little value in supervising thirty elementary students. Since the teacher is available, it seems advisable to make use of his knowledge in the system.

The tutorial side may be broken down into the elements: 1) explaining the idea, 2) providing practice to show how the idea may be used, 3) testing for acquisition of the idea by the student, 4) identifying problems that may be occurring in the acquisition, and 5) reexplaining the idea in a different manner. The cycle then starts over at #2 and repeats until successful acquisition occurs or the teacher must move on to the next major idea because of time restrictions on a course or the majority

of the students are ready to move ahead. A prime justification for any CAI system would be its ability to eliminate the movement ahead due to the latter two reasons.

The explanation of a new idea consists of introducing the vocabulary of the idea, explaining the relationships between the new idea and ideas previously taught, an indication of the context in which the idea is applicable, and examples of how the idea may be applied. The student then generalizes upon the idea so that he is able to distinguish in most of the cases when to use it.

The purpose of practice is to reinforce the student's generalizing mechanisms. The teacher must have knowledge of and supply problems to which the idea is applicable and which best demonstrate the variety of applications in which the idea may be applied.

The purpose of testing is to insure that the student has generalized upon an idea properly and has adequately grasped the relationship of the idea to other ideas. In preparing a test, the teacher must have a knowledge of questions which are discriminating, i.e., not all so easy everybody gets them right or so difficult that nobody gets them right, and which will identify the errors in generalizing and relationships the student may make.

The teacher must have worked the problem beforehand and know the answer. He can then compare his sequence of steps with those taken by the student and see where they diverge. In most cases, the sequences will diverge at points anticipated by the teacher based on prior experience

in which case he will know why the student took the incorrect step and be able to remedy the misconception. Occasionally, though, the teacher must ask the student why he took the step he did. In this case, correction is not as simple. The teacher must do some generalizing on his own to determine how this particular mistake relates to other mistakes he has encountered. Errors do not provide the only clue to student misunderstanding. The amount of time it requires for the student to answer a question is also indicative of a weakness.

Teachers, through experience, develop a "bag of tricks" which enable them to correct student misunderstanding. The choice of trick to use depends on the prior success or failure of a particular trick in correcting similar errors in the past. When more than one "trick" will apparently fill the bill, the choice as to which to use rests upon the teacher's knowledge of the student's abilities and prior experience.

As suggested in the previous discussion, the teacher does not step into his first classroom with a complete knowledge of what to do, although an outsider might obtain that impression without knowledge of the behind-the-scenes activities. Good teachers must prepare lessons in which the objective is clear and the method interesting, thorough and flexible. Each lesson should be evaluated by the teacher to see if his objective was reached, and if not, find the reason why it was not reached.

To help the teacher in preparing, the teacher's edition of a text usually states the objectives of the lesson and suggests methods by which the objectives may be met. It provides practice problems and

explains major problems that may arise and sometimes suggests the tools that may be used to correct these problems in understanding. As the teacher's experience grows, he develops his own methods as to what to do when certain errors occur. He also develops a better understanding of what problems are encountered.

B. SYSTEM RESPONSIBILITIES

It was decided that to be an effective aid to the teacher, a CAI system should be able to perform all of the functions discussed above. However, since such a system is going to be a beginning teacher, in comparison to the actual teacher, it was deemed allowable to give the teacher the responsibility of providing the same information to the system which he is given in the teacher's manual or has knowledge of from past experience. However, the amount of time and effort required of the teacher should not become burdensome, either in the time spent training the computer or learning technicalities. To keep this time at a minimum, it was necessary to add a third partner to the system, namely, a programmer. In the system visualized, the interactions and responsibilities discussed below are necessary for successful operation.

1. The Teacher's Role

Pending a major breakthrough in the field of Artificial Intelligence, the teacher must maintain the dominant role in the system. He is the storehouse of knowledge and experience to which the computer must turn for assistance.

The teacher must provide the computer with his knowledge of the kind of problems that may be encountered in the learning of a particular idea. He must, at the same time, provide the computer with some knowledge of the remedial measures to take after the problems occur. This information may either be given in direct cause and effect form, i.e., if the student makes this error then do this, or it may only be a list of possible methods to use when errors occur in the presentation of an idea. In the latter case, the computer must be able to recall which methods have previously worked in similar situations and make a selection of which method it feels will work in this case. The teacher is not infallible, thus the computer must be able to indicate when a particular trick does not work and either try one of its own or request another idea.

The teacher must indicate to the computer what questions are best suited for instilling the idea to be taught. This is not simply a matter of giving the form of the question. It must include some indication as to the conditions which determine the difficulty of the question, if the computer is to make effective use of the information contained in the student's response. Whether the conditions are specifically stated to the computer or deduced by the computer from examples is a matter of choice and programming, but they must be obtained.

It is the teacher's responsibility to see that the student is able to understand the material that is being presented. He may either do this by presenting the idea initially and familiarizing the student with the types of questions before the computer takes over. This method,

however, restricts the tutorial ability of the computer in an area where it appears it can be of use. The other alternative is to supply information to the computer on how to present the idea initially. Lesson frames may be prepared beforehand and the computer can construct examples from the problem descriptions to show the student the steps which should be followed to obtain an answer. If the computer does the presentation, one must make certain that the means it has available to make the presentation are suitable to the student's level.

The teacher must be ready to provide assistance when the computer fails in getting an idea across. To build a system that would handle all possible situations that might arise would only increase the costs disproportionately to the number of students helped by adding to the system means to handle the situations which occur infrequently. Since the computer has relieved the teacher of checking up on all students, his time should be devoted to the cases with which the computer is having trouble.

In order for the computer to interact properly with the student, the computer needs to know the student's level and foreseen capabilities. The computer will form its own conjectures as it works with the student, but this information is required at the start of the student's first lesson. It seems a simpler matter to have the teacher supply the information than for the computer to test each child and make an evaluation before beginning the first lesson. Further, the teacher may desire to modify the computer's conjectures and should have the ability to do so.

2. The Computer's Role

The prime functions assigned the computer are generating problems and presenting them to the student, identifying errors and providing proper remedial material, and deciding when a student is ready to advance.

The computer must be capable of generating questions as a function of its interaction with the student. It must adjust the difficulty of the question to the readiness of the student who is to answer it. Furthermore, the questions should be generated in a random fashion so that each student is provided with a different sequence. The type of problem it generates is the responsibility jointly of the teacher and the computer. The teacher must provide the form of the problem, but it is the job of the computer to decide if the problem is commensurate with the student's ability. Once the computer is aware of the student's level of achievement, it must decide if the problem it is preparing to ask is a hard problem or an easy problem. Most teachers begin by asking the student easy questions until he has gained sufficient confidence and then increase the level of difficulty.

There are apparently two ways to approach the generation of a question of a particular degree of difficulty once the conditions governing the difficultness are known. Either the machine can generate a problem and then apply the conditions to determine if the problem meets them. If the problem does not, it can either be stored or thrown away and another problem generated. This process continues until a problem of the right

difficulty is generated. The problem here is that in a random generation, most questions generated will be of medium difficulty. It may require several attempts to produce an easy problem or a hard problem. The second method is to have the computer decide how difficult a problem is desired and then to apply the conditions during the generation of the problem. The problem here is that applying the conditions during the generation of a question is more difficult than simply testing a question to see if it meets the conditions. It is not clear whether the extra time spent is more or less than the time spent waiting for a question of the right difficulty to occur.

Since it is necessary for the computer to generate an answer to any problem it creates, it seems logical that it be able to use this answer to identify the error made by the student. If the program produces its answer in the same manner as the student produced his, it is then in the unique position of being able to compare the student's steps with its own to determine where the student went astray. In this respect, the system is acting much like a theorem-prover, that is moving down a tree of legitimate deductions from a point where the student and the computer agreed to some new conclusion where they differ.[6] Once the problem is identified, it is a simpler matter to select a method of providing remedial assistance. Either the computer can use a method it has been told to use

---

[6] Uhr, L., 24th Conference Proceedings of the Association for Computing Machinery, "Teaching Machines that Generate Problems as a Function of Interaction with Students," p. 128, 1969.

when this occurs, or it can generalize and use a method its experience has shown to work before or that has worked in a similar situation.[7]

The mechanism for deciding when a student should advance must allow sufficient questions to be asked to insure that the student has gained enough proficiency in the idea being presented to be able to comprehend the next idea. At the same time, it must not ask too many questions, least the student become bored. If this occurs, one of the main arguments for the system--namely, its ability to make learning a more interesting experience--will be lost. The situation is further complicated by the fact that in some instances a student who was slow to catch on to the last idea because it was not presented properly or for some other reason, may catch on very rapidly to the idea now being presented. Therefore, if the mechanism for deciding when to proceed is dependent upon a student's apparent rate of comprehension, we must make sure that the method for determining this rate of comprehension is dynamic. In later grades, this problem may be reduced by allowing the student some responsibility as to the manner in which he progresses from idea to idea. In earlier grades where practice is essential, it has been suggested that this matter may be of lesser importance because generally students enjoy doing what they can do and would not become bored. However, it is felt that this mechanism must have some dynamic quality, else why bother to individualize instruction. The smarter student may be content to show his prowess and the slower student just as happy to move on to something else.

---

[7] Ibid.

3. The Programmer's Role

Educationally speaking, the role of the programmer is minor. Functionally speaking, it is of major importance. His is the responsibility of seeing to it that the files established contain all the information required for making meaningful reports to the teacher and all the information utilized by the computer in its decision making. He is also responsible for creating the computer's library of "tricks" based on the teacher's recommendations of the remedial methods required in a particular course.

4. Interface between Teacher and Programmer

The teacher has the job of initiating communications across this interface by making his needs known to the programmer. In the type of system visualized, the programmer would not be a part of arranging the learning sequences. This, hopefully, will ease the communication problem discussed earlier.

5. Interface between Teacher and Computer

In the relationship between the teacher and computer, it is the computer which must take the responsibility for successful communication if it is expected that the system will not add to the burden of the teacher.

Essentially, this means the computer must be able to communicate with the teacher in the teacher's own language. Although some effort is required in well-structured subjects such as mathematics and the sciences, this is a solvable problem since these subjects have a language all their own. Simmons discussed several natural language systems.

CARPS by Charniak and STUDENT by Borrow have application in the subject we are discussing.[8] Uhr suggests that the same may also be true in the teaching of foreign languages, although the problem is not as clear cut.[9] In subjects such as arts and history, where the language used is much greater in size and scope, the picture is even less clear.

Failure to provide a natural means for the teacher to converse with the computer requires that the computer prompt the teacher for the information it desires. As the teacher gains experience with the information that the computer needs and the format in which it is to be delivered, the prompting mechanisms will probably become an odious delay in getting the information to the computer. There will probably also be times when the teacher does not have the time to give the information to the computer directly and would rather leave the information for the computer to digest while the teacher is engaged elsewhere.

Thus, it is necessary that this channel of communication be able to act in at least three modes: 1) an interactive-prompted mode, 2) an interactive-unprompted mode, 3) an off-line mode. In each of these modes extensive error-checking routines must be available to the computer to insure that it has received all the information it requires and in the right format.

---

[8] Simmons, R. F., "Natural Language Question Answering Systems: 1969," Communications of the ACM, v. 13, p. 21, January 1970.

[9] Uhr, L., 24th Conference Proceedings of the Association for Computing Machinery, "Teaching Machines that Generate Problems as a Function of Interaction with Students," p. 126, 1969.

Furthermore, the teacher may not desire to impart a totally new set of information to the computer, but only to modify information previously given. Thus, the system also requires an extensive editing package which must meet the same requirements as the original I-O package.

IV. IMPLEMENTATION

In order to test the applicability and the veracity of the ideas discussed in Section III, construction was begun on a system which would be capable of teaching fourth grade mathematics. The system was given the name RASCAL (Rudimentary Adaptive System for Computer-Aided Learning). Rudimentary because it was to be a first try and hence, certain simplifications were made to the original ideas. Adaptive because it was hoped that the system would be able to generate questions and present remedial material based on its interaction with the student. The selection of fourth grade arithmetic as the subject to be taught was based on several factors: 1) fourth grade arithmetic is a highly structured subject with the relationships between the ideas taught well established, 2) the remedial measures used are fairly straightforward in most cases and available in the teachers' edition of the text book used by the Monterey Unified School District, 3) the descriptions of problems to be asked are fairly simple and relatively limited in format.

A. STRUCTURE OF RASCAL

Internally, RASCAL is structured into Levels which correspond roughly to a daily classroom lesson. This generally consists of introducing a new idea and practicing it, or extending the previous idea to larger numbers. Each Level is then broken down into Problem Types which represent the type of questions to be asked in cementing the ideas

corresponding to the Level. Each Problem Type consists of a description of the form of the question and a list of the conditions that describe the difficulty of the question. Questions are classified as Hard, Medium or Easy. Associated with each Problem Type is a Tree which describes the next action to be taken by the computer. These Trees constitute part of the tutorial mechanism of the system. Each node of the Tree contains the number of times a node is entered, the conditions for entering a node, and the process to be carried out if the conditions are met. The nodes are grouped together in sets based on their applicability to the node preceding them in the Tree. Figure 2 is a simplified diagram of this structure. For a complete example, refer to Appendix A.

The Problem Type for the question is selected on a random basis. Once the Problem Type has been selected, the computer examines the conditions applicable to the difficulty it desires to make the question and generates a question accordingly. Each Level has associated with it a Level Frame which constitutes the rest of the tutorial process. It consists of a written explanation of the idea being presented as it might appear in the text book and example problems. Thus, it resembles the manner in which a teacher may introduce a new idea. In RASCAL, the teacher is assigned the function of breaking the course into Levels and describing the Problem Types associated with each Level. He is also responsible for creating the Tree of remedial steps associated with each Problem Type. The main elements of the computer's job consist of generating problems, presenting the problem, calculating an answer, selecting a branch of the

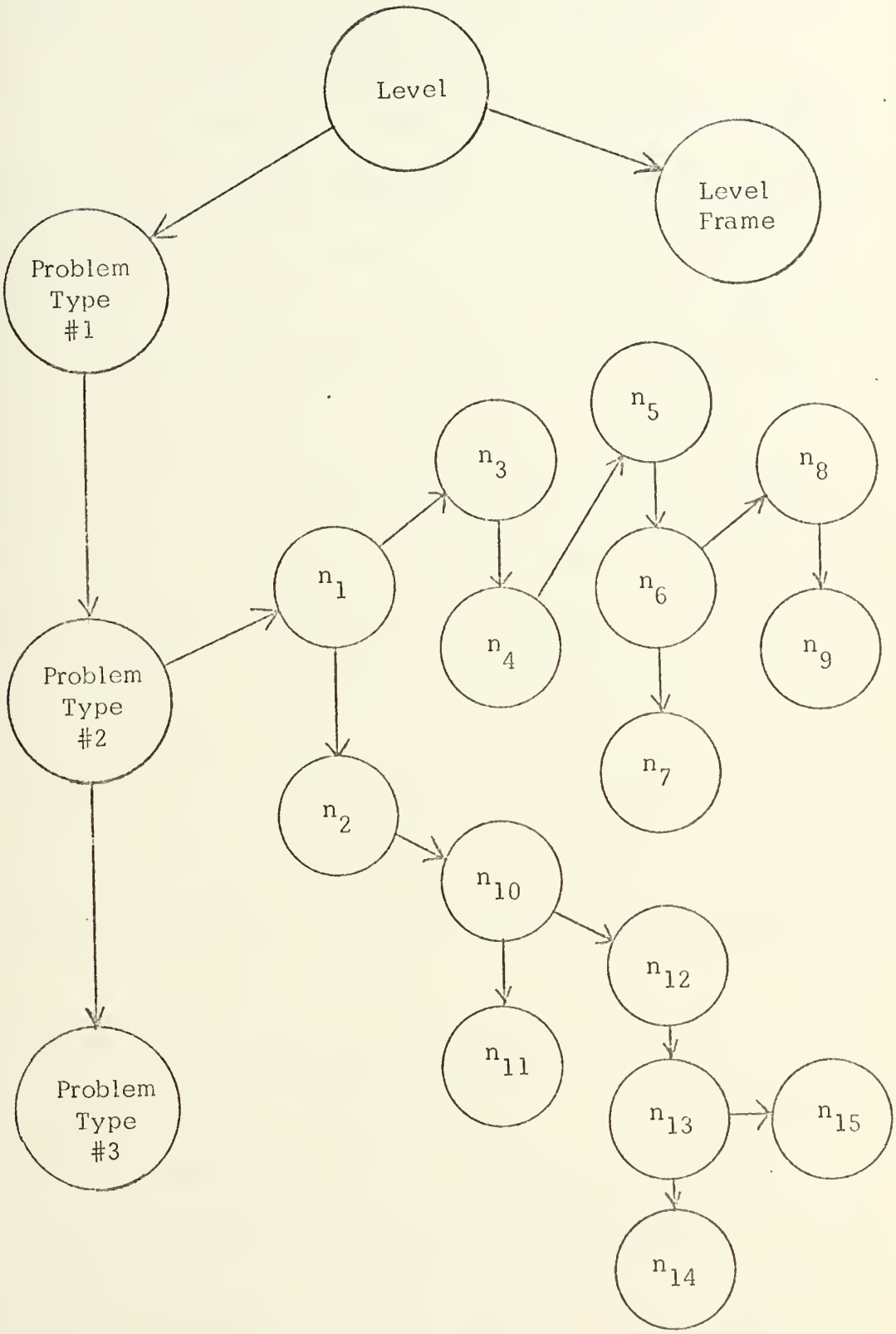

Level
Level Frame
Problem Type #1
Problem Type #2
Problem Type #3
n_1
n_2
n_3
n_4
n_5
n_6
n_7
n_8
n_9
n_{10}
n_{11}
n_{12}
n_{13}
n_{14}
n_{15}

FIGURE 2

Tree, performing the operation in the branch, and determining when to advance the student. Refer to the flow chart in Appendix A.

1. Description of a Problem Type

The description of a Problem Type consists of the format for the problem and the conditions describing each level of difficulty for the problem. The format of the problem is limited to the standard Infix form and the answer to be supplied by the student must be to the right of the equal sign. Although it is realized that fourth grade problems do not always call for the answer to be to the right of the equal sign, i.e., 7 + answer = 13, it was decided to limit the form initially and then to later expand the system to be able to handle formats where the answer to be provided could occur anywhere in the description. Arguments to fill the operand locations consist of the twenty-six alphabetic letters and operators are the four elementary mathematical operators, '+' for addition, '-' for subtraction, '*' for multiplication, and '/' for division. The occurrence of the same letter in operand positions will cause the same number to replace the operand in each of those positions. Refer to Example 1.

Example 1

If the problem format is

a + b - c = ;

then three numbers will be generated and the problem presented to the student might be

7 + 3 - 6 = ?.

However, if the format of the problem

were given as

$$a + (b-b) = \ ;$$

then only two numbers would be generated

and the problem presented to the student

might be

$$7 + (3 - 3) = ?.$$

The conditions for describing the difficulty of a problem must be entered in the order: Hard, Medium, Easy. In each case, the language used to describe the conditions is the same. The first conditions given are the sizes of the number to replace each distinct argument in the problem format, and these must be specified. In addition, optional conditions may be specified as is discussed in the next paragraph. The size of the number is specified in the form

(argument) '=' (number)

e.g., a = 2, which says every time the argument a is encountered, in the problem format, it is to be replaced by the same two-digit number.

Optional statements to specify conditions of difficulty allow for conditions to be specified: 1) on a particular digit of an argument, 2) between any two digits of different arguments having the same place value, 3) on the sum of the digits in any place value position, and 4) between any two arguments. Example 2 shows the form required for each of these conditions.

To identify the place value, the first two letters of the place value names are used if it is a single word, and the first letter from each

name if a two-word name, e.g., UN for units and TT for ten thousand. This method is used in preference to the actual names for simplification. To identify a particular digit, the place value name followed by the argument, enclosed in parentheses, in which the digit is located is required, e.g., UN(a) identifies the units digit in argument a.

The operators that may be used in specifying conditions consist of relational operators, '<' for less than, '>' for greater than, '=' for equal, '<=' for less than or equal, '>=' for greater than or equal, '**' for multiple of, and '//' for divisible by. The use of letter descriptions, i.e., 'L.T.' for less than or the actual words themselves were also considered as possibilities. They were not included since it only adds a non-essential and easily solvable complication to the system. The letter descriptive or word forms must be reduced prior to use. To attempt to use the symbols in letter form when the conditions are being applied, only slows down the generation of the numbers to fill the argument positions. The relational symbol for Not (¬) was also eliminated.

Each condition must be separated from its predecessor by a comma, and the last condition must be followed by a semi-colon. A formal definition of the language for specifying Problem Types is located in Appendix A. Appendix B contains a complete problem description.

Example 2

CONDITIONS FOR SPECIFYING THE DIFFICULTY OF A PROBLEM

a) On a particular digit of an argument.

Form: (place value) '(' (argument) ')' (relational operator) (number)

Examples: UN(a) < 7, TE(b) > 3

b) Between two digits of different arguments.

Form: (place value) '(' (argument) ')' (relational operator) (place value) ')' (argument) ')'

Examples: UN(b) > UN(a), TE(a) > TEN(b)

c) On the sum of the digits in a place value position.

Form: (place value) (relational operator) (number)

Examples: UN >= 10, UN <= 9

d) Between two arguments.

Form: (argument) (relational operator) (argument)

Examples: a // b, a > b, a ** b

2. Description of Trees

To properly allow for the description of the Trees for providing assistance, it was necessary to determine what types of remedial processes might be used in the Tree. The first thing that might be tried if the student has trouble would be to give an example and then ask an easier question. If this does not work, then one might try to use a special trick called a Function in the terminology of the system. If still unsuccessful, then a statement to the student of what he has done wrong might be in order. If the misunderstanding is basic, then a re-presentation of the lesson frame may be required. Also, it may be desirable to ask the student why he responded in the manner he did to help decide which of the remedial steps above is the most appropriate.

It was decided that the language used to specify the Tree should allow for all these possibilities. The general form

(Type) ':' (Action) ';'

was decided upon. Type consists of a two-letter description of the category discussed above to which the action belongs. These letters are the first two letters of the descriptive word identifying the category discussed above to which the Action belongs; thus, PR for a problem, QU for question, ST for statement, FR for frame, FU for a function call, and HA to halt the procedure and present the correct answer to the student if his answer is wrong. The Action part of the form depends upon the Type. Example 3 shows various possibilities.

If the type of process is PR, then there are six possible Actions that may be specified. HARD, MEDIUM and EASY designate that a new problem of the difficulty indicated should be generated using the same problem format. Refer to Example 3a. NEW designates that a problem of a different form is to be presented. It should be followed by the format of the new problem to be presented. If the arguments used in this description are the same as those used in the format of the original problem, the numbers of the original problem will be used to replace the arguments of the new problem. Refer to Example 3b. If different arguments are used, then the computer will generate new numbers, so the number of digits these numbers are to have must also be specified. Refer to Example 3c. REPEAT designates that the original problem is to be asked again. EXAMPLE indicates to the computer that it is to show the student an example problem of the same format and level of difficulty as the one he has answered incorrectly.

If the Type is QU, then the Action to be performed is the question to be asked, followed by the replies that are to be considered correct, enclosed in parentheses and separated by commas. These replies are limited to one word to simplify matching them with the student's answer. Later, they were to be of any size. Refer to Example 3d.

If the Type specified is ST, then the Action should be just the sentence or sentences to be presented.

When the Type is FU, then the Action part of the form is to be the name of the Function or the file number of the Function to be used.

Example 3

SPECIFICATIONS OF REMEDIAL METHODS

The original problem presented to the student is:

8 + 6 = ?

generated from the problem format 'a+b=;' using the conditions specified for a problem of Medium difficulty 'a=1,b=1,UN(a) >6,UN(b) >=4;'.

a) The remedial process is: 'PR:EASY' and the conditions for an Easy problem are 'a=1,b=1,UN(b)<=3;'.

The computer will present a problem similar to:

6 + 1 = ?

b) The remedial process is: 'PR:NEW,b+a=;'.

The computer will present the problem:

6 + 8 = ?

c) The remedial process is: 'PR:NEW,c-d=,c=1,d=1,c >d;'

The computer will present a problem similar to:

6 - 3 = ?

d) The remedial process is: 'QU:Do you understand the difference between addition and subtraction? (yes);'.

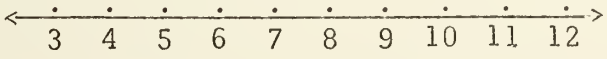
The computer will display to the student:

Do you understand the difference
between addition and subtraction?

If the student answers 'yes', the computer will consider this a correct answer in searching for the next branch in the Tree it is to take.

e) The remedial process is: 'FU:NUMBER LINE' and NUMBER LINE is a Function which presents a number line to the student.

The computer will display to the student:



The teacher, of course, will have a listing of the Functions available. However, if he has lost his copy or cannot find it, he should be allowed to call the Function by name. Therefore, the name of a Function should agree as closely as possible with the terminology of the teacher. It is still possible, though, that a match cannot be made between what the teacher calls a Function and its name in the system's library of Functions. Thus, the error-checking routines in the system should include a mechanism for determining which Function names match the closest to the name given by the teacher and determining a replacement, either by asking the teacher which one he means when in the interactive mode, or selecting the closest match and informing the teacher of the substitution when in the off-line mode. Refer to Example 3e.

When the Type is FR, the Action part of the format is the name of the level or level-number to which the frame is associated. The same comments apply to the error-checking routines as apply when the Type equals FU.

Before presenting a remedial node to a student, the computer must decide whether or not the node is applicable. Thus, each node in the Tree contains a list of conditions which describe the situation in which the node applies. The construction of these conditions is presently the responsibility of the teacher. Thus, the language for communication between the teacher and the computer provides for the list of conditions to be specified in the manner discussed below.

The conditions themselves may be based on 1) whether the answer is right or wrong, 2) the difficulty of the problem, 3) the student's ability, and 4) particular differences in the answer calculated by the computer and the student's answer. To describe a condition of the first type, the terms RIGHT and WRONG may be used. The terms HARD, MEDIUM and EASY may be used to describe the difficulty of the problem. FAST, AVERAGE and SLOW are to be used in describing the student's ability. When referring to the student's answer, the term SANS is to be used, and the term MANS is to be used in referring to the machine's answer.

It is possible to specify a particular digit of the student's answer or the computer's answer when stating a condition by the use of a period as a delimiter and a number which corresponds to the place value of that digit. This number is the location of the digit when examining the answer from right to left. Thus, 1 corresponds to the units digit, 2 corresponds to the tens digit, and so on. It is not necessary to qualify both the terms SANS and the term MANS if the digits to be compared have the same place value. If they have a different place value, then the place value must be specified in both terms. It is also possible to specify conditions on the student's answer and the answer of the computer by specifying a relation between an arithmetic combination of the student's answer and the computer's answer and a number. Qualification of the student's answer and/or the machine's answer is possible in this case also.

The set of relational operators that can be used in relating the elements of a condition list are the same as those which are used in describing a Problem Type with the addition of '&' for and, ' |' for or and ' ' for not. The hierarchy of these operators and a formal description of the language for describing Trees is located in Appendix A. Example 4 gives examples of condition lists.

While the term 'remedial' has been used to describe the branches in the Trees, this is not the only type of function the Trees serve. Nodes may also be specified for the more successful student.

3. Advancement of the Student

The mechanism for advancement of the student is dynamic in the sense that the number of problems asked in any given level is dependent upon the answers the student gives to the problems asked. Students' responses are classified into three types: 1) those answered correctly on the first try, identified as Right; 2) those answered correctly after the presentation of remedial material, identified as Prompted; and 3) those which could not be answered correctly even after the presentation of remedial material, identified as Wrong.

The first step undertaken by the computer in the presentation of material to the student is to select the Problem Type to be presented from those in the Level the student is being taught. This is done on a random basis. Once this is done, the computer knows the form of the problem. It must then decide how difficult the problem is to be. Initially, all Problem Types are given an Easy level of difficulty. For each Problem

Example 4

CONDITION LISTS FOR DESCRIBING THE APPLICABILITY OF A NODE

Assuming a hypothetical student, who is considered to be fast learner and who has answered '11' to the problem '13+8=?', which is considered to be a Hard problem, conditions of the following types might be specified:

a) The condition string: RIGHT & FAST
is false.

b) The condition string: WRONG
is true.

c) The condition string: WRONG & SLOW
is true.

d) The condition string: WRONG & (FAST |MANS.2 >SANS.2)
is true.

The condition string: WRONG & (FAST |MANS.2 >SANS)
or
WRONG & (FAST |MANS > SANS.2)
would be interpreted in the same manner.

If specified as: WRONG & FAST |MANS.2 > SANS.2
it would not be interpreted in the same manner due to the hierarchy of operators. However, the condition string is still true.

e) The condition string: WRONG & (AVE |FAST)&(MANS-SANS=10)
is false.

If specified as: WRONG & AVE |FAST & MANS-SANS=10
it is true due to the effect of the hierarchy of operators.

Type a Score is kept which indicates how well the student is doing on that Problem Type when the problems being asked are of a particular level of difficulty. Should the Score exceed or equal .7, the level of difficulty for that Problem Type is increased to the next level of difficulty, and the Score for that Problem Type is reset to zero. When the level of difficulty for a Problem Type is at the Easy level, each Right response by the student adds .2 to the Score for that Problem Type. Each Wrong response reduces the Score by .2. Those responses by the student which fall into the Prompted category have no effect on the Score of a Problem Type. When the level of difficulty for a Problem Type is set to Medium or Hard, the Score of the Problem Type is only increased or decreased by .1. When a Problem Type is set at the Hard level of difficulty and its Score equals .7, the Problem Type is marked as completed. Since Problem Types are selected randomly and since students may have more success on one Problem Type than another, it is possible that one Problem Type may be marked as completed, while the other Problem Types in that Level still require more practice by the student. Problem Types which have been marked as completed are presented only every third time selected for presentation. This provides review for the student and allows the computer to concentrate more on other Problem Types where the student has been less successful.

When all Problem Types have been marked completed, the computer checks to see if the student is ready to advance to the next Level. Each time a student's response to a question was classified as

being Wrong, that question was added to a Hard List. To test if the student is ready to advance to the next Level, the computer asks the questions stored on this Hard List. The computer keeps track of the number of problems asked and the number answered correctly. No remedial presentation or assistance is given. If the student correctly answers 70% of the problems on the Hard List, he is advanced to the next Level. If he cannot correctly answer 70%, then he is sent back to try again. When a student is returned to the same Level to try again, the level of difficulty to which all Problem Types are set is dependent upon the number of times he has failed to correctly answer 70% of the questions on the Hard List. If it is the first time he has failed, the level of difficulty on all Problem Types is set at Hard; on the second failure, they are set to Medium. If a third failure occurs, the teacher is called to identify the problem and provide assistance. After the teacher signifies that the student is ready to continue, all Problem Types are given an Easy level of difficulty and the student is allowed one more attempt. If he fails again, he is demoted to the next lower Level and the teacher notified.

All questions appearing on the Hard List, for the first two Levels below the Level at which the student is working, are stored in the student's file. They are intended as a source of review questions. How often a review problem should be asked is a pragmatic question, and it has not been decided how often this should occur in RASCAL. Some consideration has been given to the idea that it should be a parameter of the system which can be set by the teacher.

The system also allows for downward movement and demotion in a similar manner. If the Score for a particular Problem Type becomes less than minus .5, the level of difficulty for that Problem Type is reduced to the next lower Level. If the level of difficulty is already at Easy, and the Score falls below minus .5, then the Problem Type is marked. After that, a problem is presented to the student only every third time the Problem Type is selected. If the Score for a Problem Type so marked ever returns to a value greater than minus .5, the marking is removed and the Problem Type is treated just like any other Problem Type with a level of difficulty equal to Easy. This is to allow for cases where the understanding of one Problem Type may depend upon the understanding of another Problem Type. Thus, the Problem Type with which the student is having trouble is held in abeyance, until a sufficient number of problems from the Problem Type which is a prerequisite have been asked to, hopefully, allow the student to understand the Problem Type with which he is having trouble. This should not be the case too often, since Problem Types which tend to be prerequisites should be at a lower Level, but allowance must be made for the possibility. If the Score of a Problem Type ever falls below minus one, then the student is demoted one Level and the teacher notified. Each time a Problem Type is marked for falling below minus .5, a check is made to see how many others are so marked. If the percentage of Problem Types marked in this manner exceeds 25% of the number of Problem Types in the Level, a Slow student is demoted. For the Medium student the percentage is 50 and for the Fast student, 75% is the limit.

It was mentioned earlier that a third type of response, Prompted, was allowed. It is used only in the determination of the student's abilities for the next Level. The manner in which this is accomplished is that each problem asked, regardless of its category, is added to a count of the number of problems asked at that Level. Upon completion of a Level, the number of questions asked the student is compared with the average number of questions asked in the Level. The ratio of the number asked to the average number is the deciding factor. If this ratio is .5 or less, then the student's abilities are marked as Fast; if the ratio exceeds 2, then the student's abilities are marked as Slow. In between, they are marked as Average. When a student is demoted, a flag is set so when he advances again to the next Level, his abilities are marked as being one Level lower than that calculated by the above mechanism.

4. Problem Generation

Because of the manner in which the mechanism for advancing the student works, it is necessary for RASCAL to apply the conditions for describing the difficulty of a problem at the time the problem is generated. By making certain simplifying assumptions, it appears that this method will allow problems to be generated faster than the other method described earlier. Two methods for applying the conditions to the generation of the problem were considered.

The first method considered is "set reduction." The computer would produce a large set of problems and then apply the conditions to eliminate elements of the set. Then it would choose a problem at random

from those remaining after all conditions had been satisfied. This method appears to be limited by the size of the initial set the computer must generate and save. The second method might be called "trial and modify." The computer would generate a question and then begin to apply conditions to it, modifying the question so that it fit the condition. The problem with this method is that if a condition forces a change in the problem, it must then go back to check that a previously checked condition has not been violated. If the conditions are too stringent, this could become a dog-chasing-its-tail situation. The method used in RASCAL is a modification of this second method with several simplifying assumptions made.

While it is assumed that contradictory conditions are not intended, no effort is made to understand which conditions are actually meant. Minimal changes are made and if this does not remove the contradiction, it is simply ignored. Further, it is assumed that the conditions specified by the teacher will be realistic in the sense that they will not reduce the set of problems that may be asked to a very small number, and thus, become overly critical. If this assumption is made, then it is possible to eliminate the necessity for rechecking every condition whenever a condition forces a change. Instead, it is only required that one make minimal changes and be satisfied with slight deviations from the actual conditions. Allowance for the fact that the problems generated may vary slightly from the actual specifications, is the reason the student must answer 70% of the questions on the Hard List before being allowed to proceed to the next Level. This provides a check to see if he really

understands, in the event that by answering problems not up to par, the student is thought ready to advance. Example 5a gives examples of conditions that might be considered contradictory and the Action that may be used to resolve the conflict. Example 5b shows how a problem may be modified so as not to fit the conditions perfectly.

Since the generation of numbers to replace the arguments in a problem definition is done a digit at a time, it is necessary to break the optional conditions into three groups. Group 1 consists of those conditions which relate a particular digit to a number, e.g., UN(a)<6. Group 2 consists of the conditions which relate two digits, e.g., UN(a) >UN(b) or which relate the sum of the digits in a particular place value to a number, e.g., UN >=10. In group 3 are those conditions which relate two arguments, e.g., a b. To speed up the scan of the conditions and to insure that the interpreter will find all conditions, it is necessary that all conditions in Group 1 precede those in Group 2. Those conditions in Group 2 must precede those in Group 3. Refer to Example 5c.

The computer begins by picking the first argument from a list that has been created. This list contains all the arguments that occur in the problem format. It then determines the size of that argument and allocates storage to hold the number which will replace the argument. This is done for each argument on the list of arguments. When all storage has been allocated, the computer returns to the head of the list. Beginning with the Units position, the computer scans the list of conditions for all conditions belonging to Group 1 which affect the Units position of the

first argument on the argument list. If none are found, the Units position for that argument is filled with a random number in the range 0-9. If conditions are found, they are used to modify the range of the random number and then the Units position is filled. At the same time, a choice parameter is set which tells how the digit just generated may be modified when attempting to meet a condition belonging to Group 2.

Once all the Units positions for arguments on the argument list have been filled, the computer then checks for conditions on the Units position belonging to Group 2. They cannot be checked prior to this since a digit affected by a condition belonging to Group 2 may not yet have been generated. In attempting to satisfy a condition for Group 2, the computer makes only those adjustments which are allowed by the choice parameter for the digit in question. If after making all allowed modifications, the condition from Group 2 is not satisfied, it is ignored. When all conditions from Group 2 affecting the Units position have been satisfied, the computer starts at the head of the argument list and begins filling in the Tens digits of those arguments declared to have two or more digits. Those arguments having a lesser number of digits than the place value being filled are skipped over. The process continues until all the digits of the largest argument are filled.

When all the digits of all the arguments have been filled, the computer scans the list for any conditions from Group 3. In satisfying these conditions, the computer disregards all previous conditions but only minimal changes are made to satisfy the conditions in Group 3.

Example 5c gives a detailed description of how the numbers for a problem are generated.

The choice of how a digit may be modified by a condition from Group 2 is dependent upon the conditions from Group 1 encountered when generating that digit. If no conditions belonging to Group 1 are encountered, then the digit may be modified in any manner. If the conditions encountered include the use of the relational operator '//', '**', or '=', which are given top priority, no change is allowed. If the conditions use '>' or '>=', then the digit may be increased any amount providing it does not exceed 9. If the conditions use '<' or '<=', then the digit may be reduced by any amount provided it does not go below zero. If both '>' or '>=' and '<' or '<=' are used in the conditions encountered, the digit may be increased or decreased by a minimal amount.

Whenever a condition has been interpreted by the computer, it is removed from the condition list. This increases the speed of interpretation but does not allow for conditions to be rechecked after a modification of the numbers due to another condition. This is as intended; however, it does require that the order in which the conditions belonging to Group 2 or Group 3 are listed be given some thought. The computer handles these conditions as encountered so conditions that operate on the same argument must be listed in the manner which will produce the effect closest to that actually desired. Refer to Example 5d.

Example 5

GENERATING PROBLEMS OF A SPECIFIED DIFFICULTY

a. Specifications Are Contradictory

Problem: a + b = ;

Conditions: a = 1, b = 1, UN(a) > 6, UN(b) > 4, UN < 10;

If the original numbers generated to fill the Units of 'a' and 'b' are 7 and 5, then following the minimal change policy, 'a' would be reduced by one to the value of 6. Since the condition UN < 10 is still not satisfied, 'b' would be reduced by one to the value of 4. The condition UN < 10 still not being met, 'a' would be reduced by one again. This yields the values 5 and 4 for the Units of 'a' and 'b' respectively.

In this case, the minimum change policy is not the best to follow, but it does resolve the conflict. Things turn out this way because UN < 10 has priority since it belongs to Group 2.

b. Problem Is Slightly Altered from the Specifications

Problem: a/b = ;

Conditions: a = 1, b = 1, UN(a) < 8, UN(b), > =3, a//b;

With these conditions, 'a' might be 6 and 'b' could be 9. Since a//b has priority by virtue of belonging to Group 3 and because it contains the "divisible by" operator, the policy of minimum change would produce a=9, b=9 which is a slight bending of the conditions.

c. A Complete Example

Problem: a + b + c = ;

Conditions: a=2, b=2, c=1, UN(a)>4, UN(b)<5, UN(a)>UN(c), UN>10, TE<=9;

Argument List: a, b, c

1. The first step by the computer is to allocate storage for 'a', 'b' and 'c'. When this is done the condition list has been reduced to:

UN(a)>4, UN(b)<5, UN(a)>UN(c), UN>10, TE<=9;

2. The condition list is then searched for conditions on the Units position of 'a' and a match is made on UN(a)>4. Thus, a random number is generated in the range 4-9. Suppose 5 is the number selected. It is indicated that this number can be increased. This leaves the condition list:

UN(b)<5, UN(a)>UN(c), UN>10, TE<=9;

3. The condition list is searched for conditions on the Units position of 'b' and a match is made on UN(b)<5. Thus, a random number is generated in the range 0-4. Suppose 2 is selected. It is indicated that this number can be reduced. This leaves the condition list as:

UN(a)>UN(c), UN>10, TE<=9;

4. The condition list is searched for conditions on the Units position of 'c'. No match is found so a random number is generated in the range 0-9. Suppose 6 is selected. This indicates that this number may be changed in any manner. The condition list is left as:

UN(a)>UN(c), UN>10, TE<=9;

5. Since the Units digit for all arguments has been generated, the condition list is searched for any other conditions on the Units position. The first selected is UN(a) >UN(c). Since 'a' is less than 'c', the computer must make an adjustment. The manner in which 'a' may be modified is checked and it is seen it can be increased any amount as long as it remains less than 9. The minimum change to satisfy the condition is 2 and the maximum allowed is 4. Thus, a random number is selected in the range 2-4. Suppose 3 is selected. This number is added to the Units digit of 'a', making this digit 8. The condition list is now:

UN>10, TE<=9;

6. The condition list is searched for any more conditions on the Units position and UN >10 is found. Since the sum of the digits in the Units position is already greater than 10, no action is required. If the sum was less than 10, each argument's Units digit would be examined to see if it could be increased. If a digit could be increased, a random number would be generated and the digit increased by that amount. If the Units were still less than 10, another digit would be looked for to increase. If no others could be found, the same digit would be increased again until it could not be increased further. If the UN <10 condition were still not satisfied, it would be ignored. The condition list would now be:

TE<=9;

7. Again the condition list is searched for conditions on the Units position. Since none would be found, the computer would begin to

search for conditions on the Tens digit for argument 'a'. None would be found, so a random number in the range 0-9 would be selected. Suppose 7 is selected; the indication is made to show that this number can be adjusted in any manner. The condition list remains:

TE <=9;

8. A search is now made for conditions on the Tens digit of the argument. Since none are found, a random number in the range 0-9 is selected. Suppose 9 is selected; the indication is made that this number may be adjusted in any manner. The condition list remains:

TE <=9;

9. Since the size of argument 'c' is only 1, no search is made for conditions affecting the Tens digit of argument 'c'. Instead, the search is made for any conditions affecting the Tens position and TE <=9 is found.

10. Up to this point, the numbers generated are:

a = 78, b = 92, c = 6.

Thus, the sum of the Tens digits including the carry is 17, which is greater than 9. The Tens digit for argument 'a' may be adjusted downward. However, a net change of 8 is required and the Tens digit of 'a' may only be reduced by 6. Thus, a random number in the range 1-6 is selected. Suppose 4 is selected, and the Tens position of 'a' is reduced by 4. Thus a = 38, and the sum of the Tens digits is now 13. Since the sum is still too large, the Tens digit of argument 'b' is examined to see if it may be reduced. Since it can, and since it may be reduced by 8

while the reduction required is only 4, a random number in the range 4-8 is selected. Suppose 7 is selected. This number is then used to reduce the Tens digit of argument 'b'. Thus, b = 22.

11. Since all conditions have now been satisfied, the search ceases and the problem presented to the student is:

38 + 22 + 6 = ?

12. Had the condition list been incorrectly prepared so that all the conditions of Group 1 did not precede those of Group 2, i.e., a=1, b=2, c=1, UN(a) >UN(c), UN(a) > 4, UN(b) <5, UN> 10, TE<=9;, the following would have occurred:

The search for conditions on the Units of 'a' would have yielded UN(a) >UN(c), which does not qualify as a condition belonging to Group 1, and would have been ignored. This is because the search is geared to find the first occurrence of 'UN(a) >'.

When the time came to search for conditions belonging to Group 2, the condition string would be:

UN(a) >UN(c), UN(a) >4, UN >10, TE<=9;

The condition, UN(a) >4, would be found but, not belonging in Group 2, it would be ignored. Thus, the fact that the Units of 'a' should be greater than 4 would never be established.

d. Importance of the Order of Conditions in Group 2 and Group 3

Problem: (a+b) + (a-b) = ;

Conditions: 'a' is 1 digit, 'b' is 1 digit, and UN(a) >UN(b), UN <10 are desired.

The condition string could be written as:

1) a=1, b=1, UN(a) >UN(b), UN <10;

or

2) a=1, b=1, UN <10, UN(a) > UN(b);

Form 1 should be used if it is more important that the condition, UN <10, be true than for the condition, UN(a) >UN(b), to be true, since this insures that the Units will be less than 10. There is no guarantee that the Units of 'a' will be greater than the Units of 'b'. Consider the case where a=8, b=4, after applying the condition UN(a) > UN(b). To satisfy the condition, UN <10, 'a' will be reduced by a number in the range 3-7; thus, it is possible that the result may turn out to be a=2, b=4.

Form 2 should be used if it is more important that the condition, UN(a) >UN(b), be true than for the condition, UN <10, to be true, since this guarantees that the Units position in the answer will be less than 10.

If both are equally important, then the condition should be written as:

UN(a) <=5, UN(a) >UN(b).

This insures that both conditions will be met, but reduces the set of problems that can be generated.

5. Generating an Answer

Once the numbers have been generated to replace the arguments in a problem, the computer must calculate the answer to the problem. Since the computer is well known for its calculating abilities, it would be a simple matter for it to go through the problem format, performing the indicated operations. However, this would not provide information as to the steps involved for future comparison with the steps taken by the student. Therefore, a more sophisticated process is employed.

The original problem in Infix notation is converted to Polish notation. After each operation is performed, the result is stored in a chain of partial answers, thus providing a sequence of answers which can be compared with the student's answer.

In addition, each operation is performed in the manner it would be expected that the student would perform the operation. Addition and subtraction are done digit by digit with borrows and carries remembered. Multiplication is done so that each sub-multiple is retained and these are added to produce the final answer. Division is performed by comparing the divisor with the digits of the dividend from left to right until a division resulting in a number greater than zero can be performed. The division is then performed on that partial dividend, and the remainder determined. The process is repeated on the remainder and the remaining part of the dividend.

6. Interpreting Trees

The interpretation of trees consists of two parts: interpreting the conditions for a branch, and interpreting the process to be performed in that branch. Interpreting conditions is done by converting the list of conditions into Polish notation and then operating on the Polish form by using a pushdown stack. Thus, the interpreter is not as fast as it could be. It must consider the whole condition string before arriving at a truth value for the string. For example, the condition string 'RIGHT & FAST' is converted into the Polish string 'RIGHT, FAST, &'. This forces the examination of the truth value of the term FAST even if the student's answer is wrong. A faster method would be to group & and | operations and test them one at a time, stopping as soon as one element of the And condition is false or as soon as one element of an Or condition is true.

The placement of the nodes during the interpretation is critical. After completing the process at a node, the computer examines the conditions on the set of nodes at the next lower level in the Tree, associated with the node just completed. These nodes are linked together in a sequence. The examination starts at the first node in the sequence and stops as soon as a condition string is found whose value is true. Consider Examples 4b, 4c, 4d. All of these have a value which is true. Thus, if they are in the sequence 4b, 4c, 4d, the condition strings 4c, 4d will never be checked. Since 4c is less stringent than 4d, it should not precede 4d. Else 4d may not be examined as often as it should. The correct ordering is 4d, 4c, 4b.

Once the condition interpreter has found the proper node, then the process associated with the node must be carried out. This interpretation is geared to the keys discussed on page 41. Once the key has been identified, it is only necessary to further qualify the action in some cases and then perform the action associated with the key.

B. DEVELOPMENT

It was decided to use Programming Language/One (PL/I) to implement the workings described in Section A. The choice was governed by the prime necessity of using a language with string manipulation capabilities and which could be used in an On-line situation. This narrowed the choice to SNOBOL IV and PI/I. While the string manipulation facilities of SNOBOL outweigh those of PI/I, PL/I was chosen because of other limiting factors on the use of SNOBOL.

Having reached the programming stage, it became apparent that an organized sequence of developmental steps was necessary due to the size of the program planned. The first stage was to be the development of a Drill and Practice system built around Levels and Problem Types, and incorporating the advancement mechanism described earlier. Input of the required information to construct the Levels and Problem Types was to be non-interactive at this stage, the information being obtained from preconstructed files. The Levels were to comprise only addition, subtraction, multiplication, and division problems dealing with integer numbers.

At stage two, the addition of the Trees describing remedial material was to be accomplished. This included the construction of an interpreter for conditions and processes, and the construction of the routines for carrying out the processes. It also requires the construction of the routines which perform the more common tricks associated with problems that the system was limited to at this stage.

Stage three was to be the construction of student files and to test out the system on actual fourth grade students. Stage four was to be the creation of the teacher's interactive side of the system. This included construction of the prompted I-O package, error-checking routines and editing routines, and the output to the teacher of results with students.

At stage five, the system was to be expanded to handle problems defined in other formats than the one discussed on page 37 , and expansion of the Levels to include fractions and decimal numbers.

At stage six, the main idea was to make those modifications which it was felt would make the system run more smoothly based on the results obtained to this point.

Stage seven was to begin making extensions to the system as discussed in Section V.

V. CONCLUDING REMARKS

To date, stage one has been completed and some programming has been done on stage two. The results of this work, while incomplete, do indicate that the system is feasible, and with some modifications and extensions, can be a viable means for providing individualized instruction.

Operating as a drill and practice type system with all routines in core, RASCAL used approximately 100K of storage. Thus, it is estimated that with the routines for handling Trees included, RASCAL could be run on a system with 250K of core. Since some overlaying is possible, this figure is probably overly pessimistic. Any overlaying done must be carefully planned so as not to affect the interactiveness of the system. This, especially, is a factor since the attention span of most elementary school children is shorter than that of adults. While, at present, RASCAL appears to be interactive enough, the effect of the modifications to be discussed is unknown. Since the size of the computer systems now being discussed for CAI is quite large, it does not seem unreasonable that they should have some multi-processing capability. This would insure that systems such as RASCAL would be interactive enough, since while one problem was being asked, the next problem to be asked could be generated. The size of these computer systems also makes the core requirements of RASCAL satisfactory.

For reasonable conditions on the difficulty of a problem, RASCAL can, by eliminating the rechecking of a condition, generate problems

fast enough to prevent the student from having to wait between problems. However, two failings in the present method of generating problems have been noted. The first is that the failure to recheck a condition, when generating the problem, makes it necessary for the teacher to spend more time in constructing the conditions to insure they are in the proper sequence. The second is that another group of conditions appears to be needed for complicated problems. For example, the problem '(a+b)/c= ;' might require the condition, '(a+b)//c', to be specified. The first problem may be solved either by modifying the method to include a recheck of conditions or by making the computer do the necessary ordering. The second problem requires only an expansion of the interpreter which interprets conditions regarding problem difficulty.

One further comment regarding the problem generator, deals with the method of identifying a particular digit of an argument. It was decided to use 'UN(a)' to describe the Units position of argument 'a' instead of 'a.1' because of the intent to expand the system so that Units (a) would be allowable. This was thought to be clearer to the teacher. Such thinking was not carried over to the interpreter for determining when a node was applicable where the '.1' notation is used to qualify a reference to the machine's answer or the student's answer. This inconsistency may be a source of confusion necessitating the change of one method.

From the teacher's standpoint, the method for describing a Problem Type appears reasonable. The examples in Appendix B were written with little difficulty by a fourth grade teacher after a brief explanation of the

constraints in the language. An ideal extension would be for the computer to be able to interpret the word description of the conditions which precede the actual specifications in Appendix B.

The specification of the Trees to be used by RASCAL is considerably harder, and as presently written, would probably not be acceptable to the teacher. There are two major reasons for this unacceptability. The necessity of properly ordering the nodes, as discussed earlier, greatly increases the amount of effort that must be expended. This could be corrected, either by providing a means for the computer to order the conditions, or by modifying the present means of determining the applicable nodes. This new method would allow the computer to test all nodes for applicability.

The second reason is more basic to the concepts of the system. The teacher uses his "bag of tricks" almost without thinking. Through experience, the correct method occurs to him almost as a reflex action. Thus, it is difficult for him to sit down and put on paper what may occur when a particular problem is presented to the student. Even when he does make this effort, he finds himself blocked by the limitations in the language which only adds to his frustration. Thus, it may have been premature to eliminate the programmer from the construction of the Trees. It may be necessary to have the programmer convert the English descriptions of a node, similar to those in Appendix B, into tasks that the computer can perform. This should still be an easier job than trying to create a set of frames to present to the student.

Two elements in the present system also need to be corrected to make the Trees more flexible. At present, the Trees are true trees with no loops allowed. However, certain types of errors require the same procedures to be followed; thus, it is wasteful of core, to say nothing of the burden to the teacher, by forcing repetition, not to allow looping. However, if looping is to be allowed, two problems must be taken care of.

First of all, the computer must have some means of insuring that students do not get caught in a loop. Secondly, there must be a means for the teacher to specify which node starts the procedure he wants to use at a particular point in the Tree. For the teacher to say 'Go to Node 5,' requires that he have an understanding of the system and how it numbers the nodes. Since this is one of the things one does not wish to require of the teacher, it lends support to the idea of having the programmer construct the Trees. A possible method of doing this, without using the programmer, would be to use a terminal with graphic capabilities. Then the computer could display the Tree to the teacher, and he would only have to point to the node he wanted. The computer would handle the linking. Such a terminal would also be useful in the Edit routines required by the system for the same reason.

The second correction required is that the computer needs to check to see if the student has performed the correct operation. If the problem were 'a*b= ;', the student, not really understanding multiplication, might add. The simple way of providing this capability would be to add conditions of the form, 'SANS=a+b', to the types of conditions allowed

for specifying the applicability of a node. A better way might be to have the computer automatically do this check; and, if the situation arises, branch to a routine for handling the occurrence. This would be a start on extending RASCAL so that it creates its own Trees from a set of general methods specified by the teacher. This would be a difficult task to accomplish, but it would alleviate the problems encountered in having the teacher create the Trees.

The idea that RASCAL should be able to follow the steps the student takes in reaching his answer has been partially implemented in that the system performs arithmetic operations the way the student would and stores the partial results. The mechanisms for actually comparing these to the student's steps remains to be implemented and should be given a high priority in completing the implementation of the system.

A final extension to any system such as RASCAL would be for the system not only to generate its own Trees, but also for it to generate its own remedial methods. This is an extremely difficult problem and is similar in concept to the idea in Samuel's Checker-Playing program, where the computer would generate its own parameters by which to evaluate its moves. Concrete ideas on how to do this do not exist, and it is likely that it will be some time before there is a CAI system which can generate its own methods for providing assistance to the student.

APPENDIX A

RASCAL - THE COMPUTER'S SIDE

FORMAT DESCRIPTION OF LANGUAGE FOR COMMUNICATING WITH RASCAL

The meta-symbols used in the description of the language serve the following functions:

< >	are used to enclose items which are not elements of the language and can be broken down further.
[ ]	are used to enclose optional items.
{ }	are used to indicate a choice of one of the items enclosed is to be used.
|	is used to indicate the 'Or' operation.
' '	are used to contain word descriptions of items which cannot be defined in another manner. In the event there is a possibility of confusion between a metasymbol and a symbol in the language, the actual symbol is also enclosed in quotes.
$>_{1}^{10}$	is used to indicate a limitation on the size of the items enclosed.
::=	is used to indicate a definition.

<language>::=<problem description><tree description>

⟨problem description⟩::=<problem format><hard conditions><medium conditions><easy conditions>

⟨problem format⟩::=<operand>⟨arith operator⟩<operand>=;

<operand>::=<argument>|<argument>⟨arith operator⟩<operand>

<argument>::= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z

<arith op>::= +|-|*|/|

<hard condition>::=<condition list type 1>;

<medium condition>::=<condition list type 1>;

<easy condition>::=<condition list type 1>;

⟨condition list type 1⟩::=<digit sizes><optional conditions>

<digit sizes>::=<argument>=<number>|<digit sizes>,<argument>=<number>

<number> ::=<digit>|<number><digit>

<digit>::=0|1|2|3|4|5|6|7|8|9

⟨optional conditions⟩ ::=[<group 1>][<group 2>][<group 3>]

<group 1>::=<place value><rel. operator><number>|<group>1 ,
<place value><rel. operator><number>

<rel. operator>::= <|<=|>|>=|=|//|**

<place value> ::= UN|TE|HU|TH|TT|HT|MI|TM|HM|BI

<group 2> ::=<element a>|<element b>|<group 2>,<element a>
|<group 2> , <element b>

<element a> ::=<place value> (<argument>) <rel. operator>
<place value> (<argument>)

<element b> ::=<place value><rel. operator><number>

<group 3>::=<argument><rel. operator><argument>|<group 3> ,
<argument> <rel. operator><argument>

<tree description>::=<condition list type 2> ; <process> ;

⟨condition list type 2⟩::= [¬] <condition 1>|[¬]<condition 2>|[¬]
<condition 3>|
<condition list type 2><relational>
<condition 1>|
<condition list type 2><relational>
<condition 2>|
<condition list type 2><relational>
<condition 3>

<relational>::=&|'|'

<condition 1>::=<key word>

<key word>::= RIGHT|WRONG|FAST|AVERAGE|SLOW|HARD|MEDIUM|EASY

<condition 2>::={<machines answer>, <students answer>} [¬] <rel. operator>
{<students answer>, <machines answer>}

<condition 3>::={<machines answer>, <students answer>} <arith op.> {<students, <machines
answer>, answer>} [¬] <rel. operator><number>

<machines answer>::= MANS|MANS.<number>$_{1}^{10}$.

<students answer>::= SANS|SANS.<number>$_{1}^{10}$.

<process>::=<proc 1><proc 2><proc 3><proc 4><proc 5><proc 6>

<proc 1>::= PR:<action 1>

<action 1>::= NEW , <problem format>,<digit sizes>|HARD|
MEDIUM|EASY|REPEAT|EXAMPLE

<proc 2>::= QU:<question> (<answer list>)

<question>::='An English language question.'

<answer list>::=<answer>|<answer list> , <answer>

<answer>::= 'One word correct answer to the question.'

<proc 3>::= ST: <statement>

<statement ::= 'English sentence or sentences.'

<proc 4>::= FU: <function name>| FU:<function number>

<function name>::= 'English name of a system function.'

<function number>::=<number>

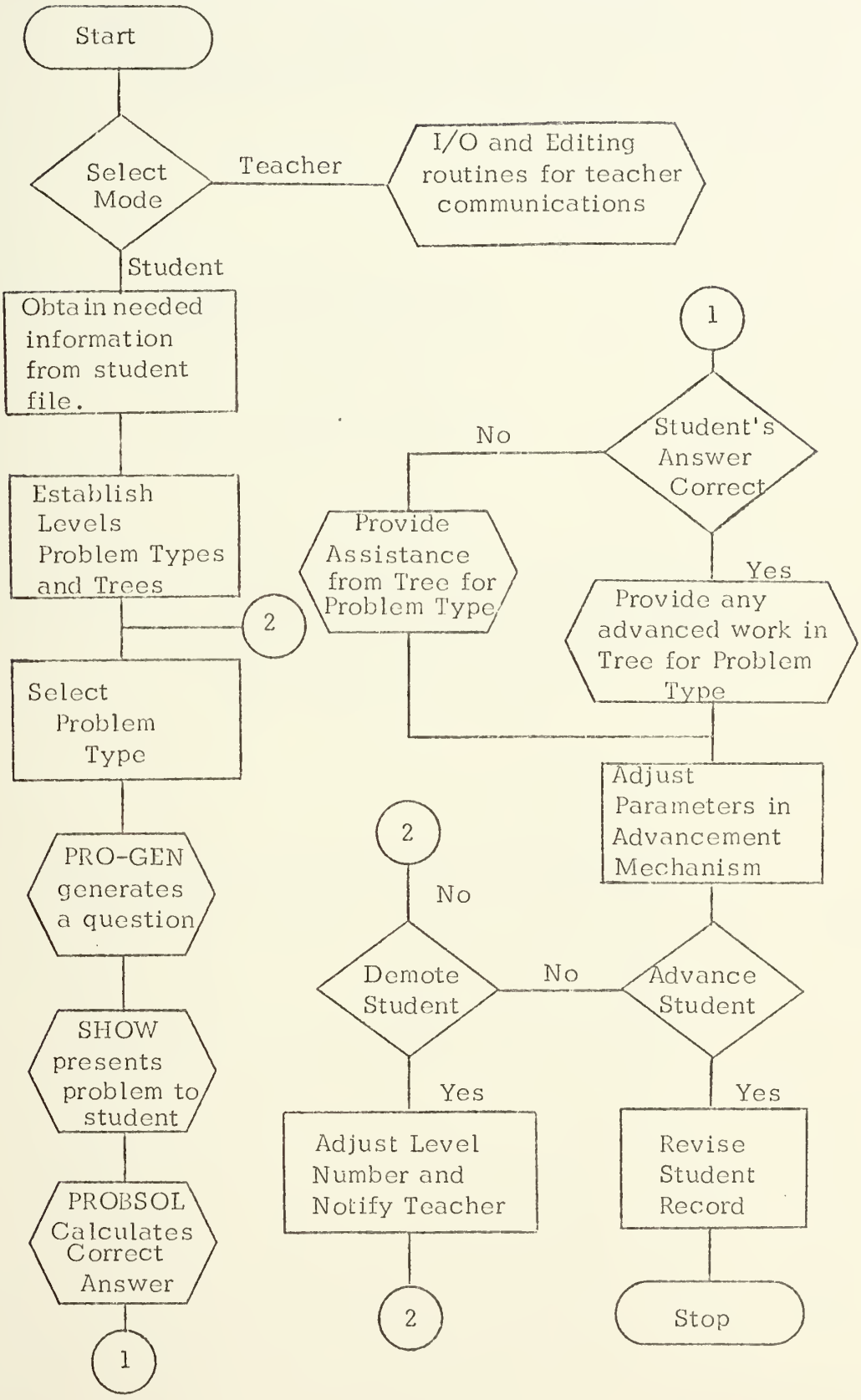
<proc 5>::= FR:<frame name>| FR: <frame number>

<frame name>::= 'English name of a system frame.'

<frame number>::=<number>

<proc 6>::= HA:

GENERALIZED FLOW CHART FOR RASCAL



HIERARCHY OF OPERATORS FOR CONVERTING

INFIX PROBLEM FORMAT TO POLISH

0	1	2	3
)	(	+	/
	;	-	*

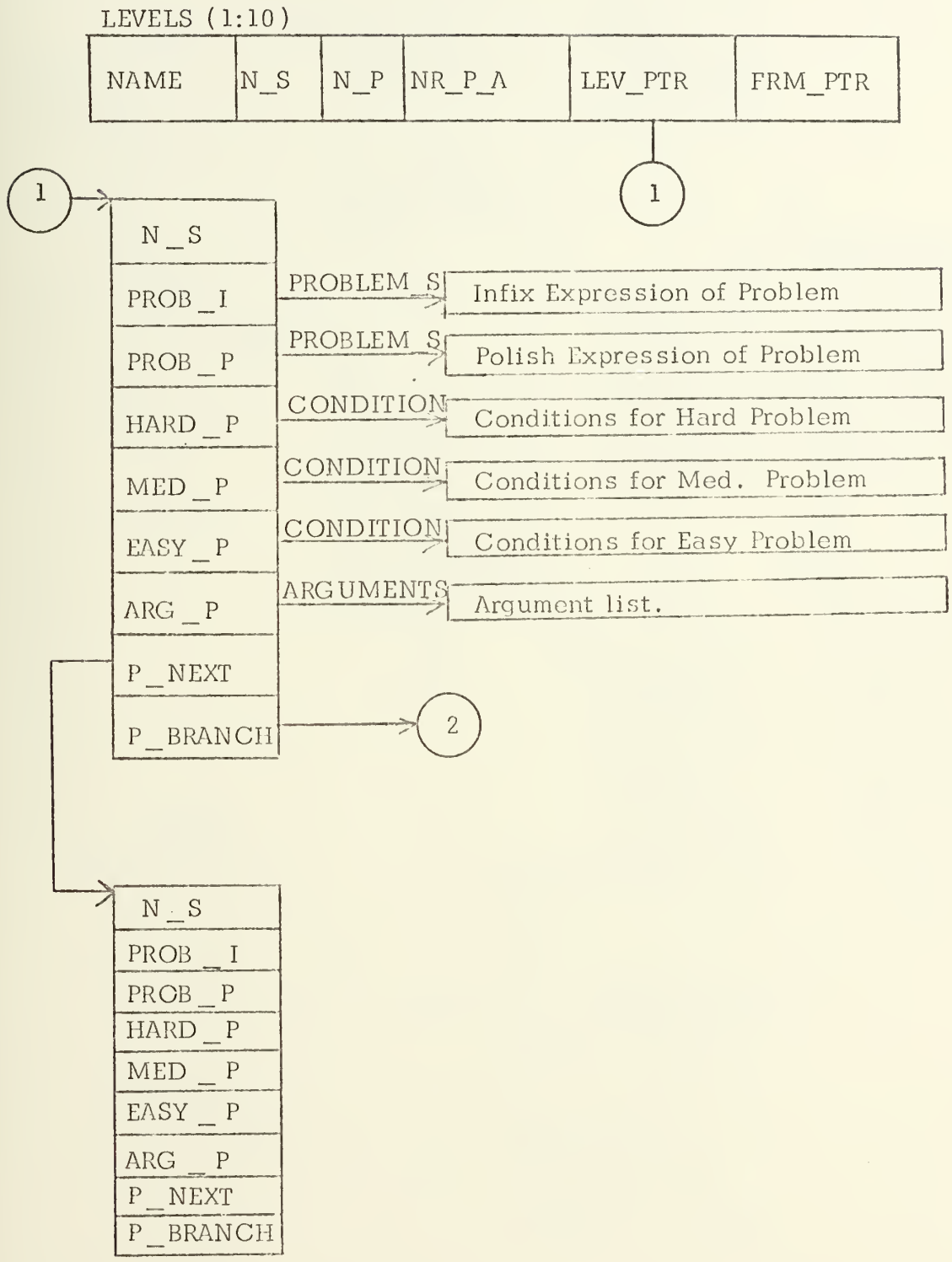
HIERARCHY OF OPERATORS USED IN CONVERTING

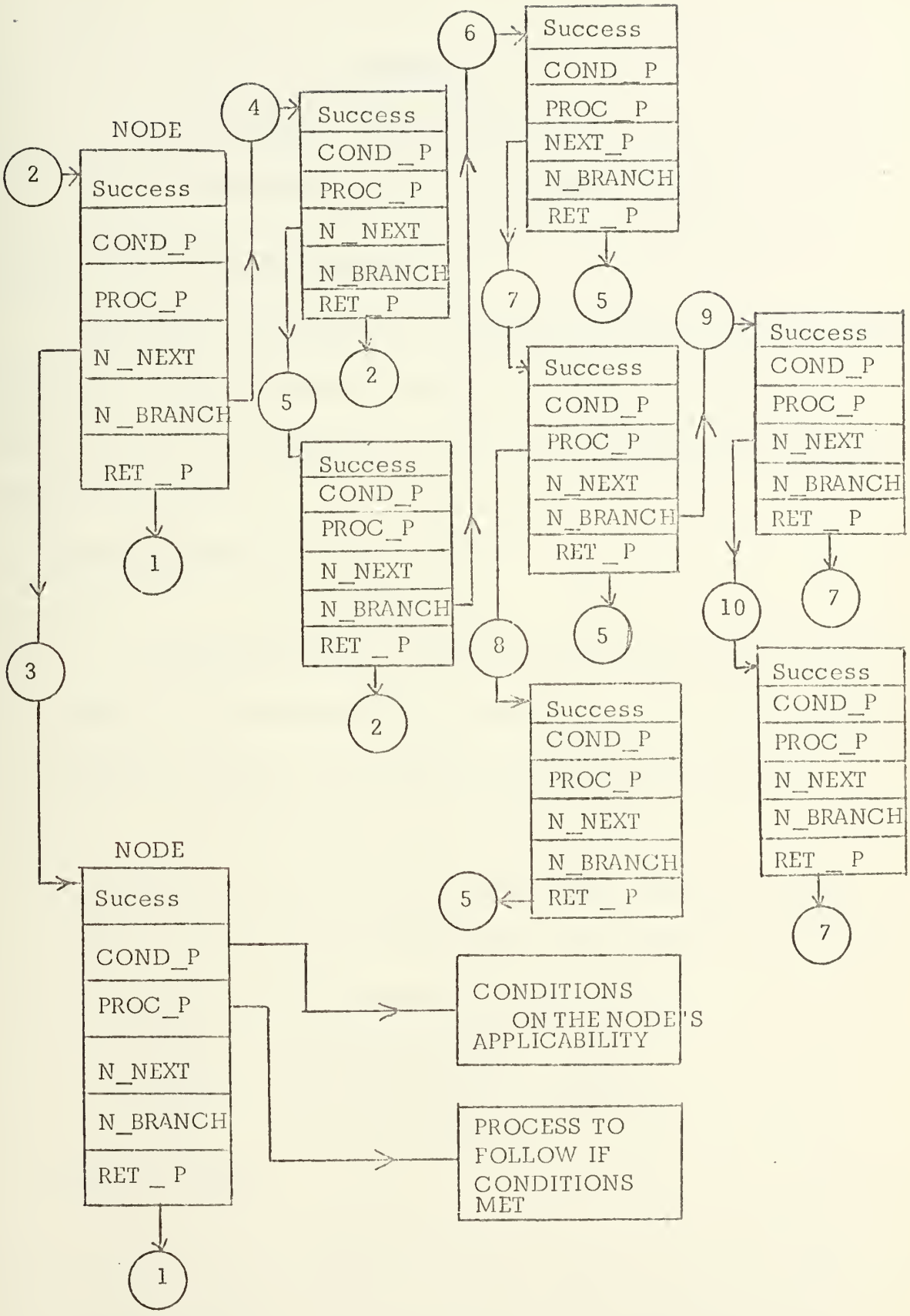
CONDITION LISTS FOR SELECTING

BRANCHES INTO POLISH

0	1	2	3	4	5	6	7
(	)	|	&	< >	+	*	
	;			<=>= =	-	/	

IMPORTANT ELEMENTS IN THE INTERNAL STRUCTURE OF RASCAL




NODE
Success
COND_P
PROC_P
N_NEXT
N_BRANCH
RET_P
NODE
Sucess
COND_P
PROC_P
N_NEXT
N_BRANCH
RET_P
CONDITIONS ON THE NODE'S APPLICABILITY
PROCESS TO FOLLOW IF CONDITIONS MET
1
2
3
4
5
6
7
8
9
10

APPENDIX B

RASCAL - THE TEACHER'S SIDE

Sample Descriptions of Levels and Problem Formats

Level 1 - ADDITION AND SUBTRACTION

The purpose of this level is to provide practice for the understanding of: 1) the relationship between addition and subtraction, 2) the basic facts in addition and subtraction, and 3) the communative and associative properties of addition.

Problem Type 1:

a + b = ;

This Problem Type is intended to provide practice in the basic assition and subtraction facts, stressing the communative property of addition and the relationship of addition and subtraction.

Problems are HARD if one number is greater than 10. Problems are considered MEDIUM if both the numbers are greater than 4. Problems are considered EASY if one number is less than 4. These conditions are specified to the computer as:

HARD: a=2, b=1, UN >10

MEDIUM: a=1, b=1, UN(a) >4, UN(b) >4

EASY: a=1, b=1, UN(b) >4

Problem Type 2:

a + b + c = ;

This Problem Type is intended to provide practice in adding more than two numbers in a problem. The communative and

associative properties of addition are stressed. Some practice in carrying in addition is given.

Problems are considered HARD if one number is greater than ten and carrying is required in the Units place. MEDIUM problems consist of three numbers less than 9. Problems are considered EASY if two of the numbers from 0-9 are under 5. These conditions are specified to the computer as:

HARD: a=2, b=1, c=1, UN>10

MEDIUM: a=1, b=1, c=1

EASY: a=1, b=1, c=1, UN(a)<5, UN(b)<5

Level 2 - ADDITION AND SUBTRACTION

The purpose of the level is to provide practice for the understanding of: 1) the relationship between addition and subtraction, 2) the commutative and associative properties of addition, 3) carrying in addition, 4) borrowing in subtraction, and 5) working with larger numbers which requires greater accuracy.

Problem Type 1:

a + b = ;

This type of problem is intended to provide practice in adding two and three-digit numbers and carrying.

HARD problems are numbers in the hundreds where carrying is required in the Units and Tens place. In MEDIUM problems, there is carrying in the Units place only. MEDIUM and EASY problems consist of numbers between 10 and 99. EASY problems require no carrying. These conditions are specified to the computer as:

HARD: a=3, b=3, UN>10, TE>10

MEDIUM: a=2, b=2, UN>10, TE<10

EASY: a=2, b=2, UN<10, TE<10

Problem Type 2:

c - b = ;

This Problem Type is intended to provide practice in borrowing in subtraction and working with larger numbers.

HARD problems, consisting of numbers between 100 and 9999, require borrowing in the Units and Tens place. MEDIUM problems, numbers 10 and 99, also require borrowing. EASY problems require no borrowing and are numbers between 10 and 99. These conditions are specified to the computer as:

HARD: c=4, b=3, c>b, UN(b)>UN(c), TE(b)>TE(a)

MEDIUM: c=2, b=2, c>b, UN(b)>UN(c)

EASY: c=2, b=2, c>b, UN(b)<UN(c)

Problem Type 3:

a + b + c = ;

This Problem Type is intended to provide practice in adding three larger numbers, carrying, and understanding the communative and associative properties of addition.

All problems require carrying in the Units, Tens and Hundreds places. They differ in that HARD problems consist of numbers from 100-999; MEDIUM, from 10-99; and EASY, from 1-99, where one number is less than 10. These conditions are specified to the computer as:

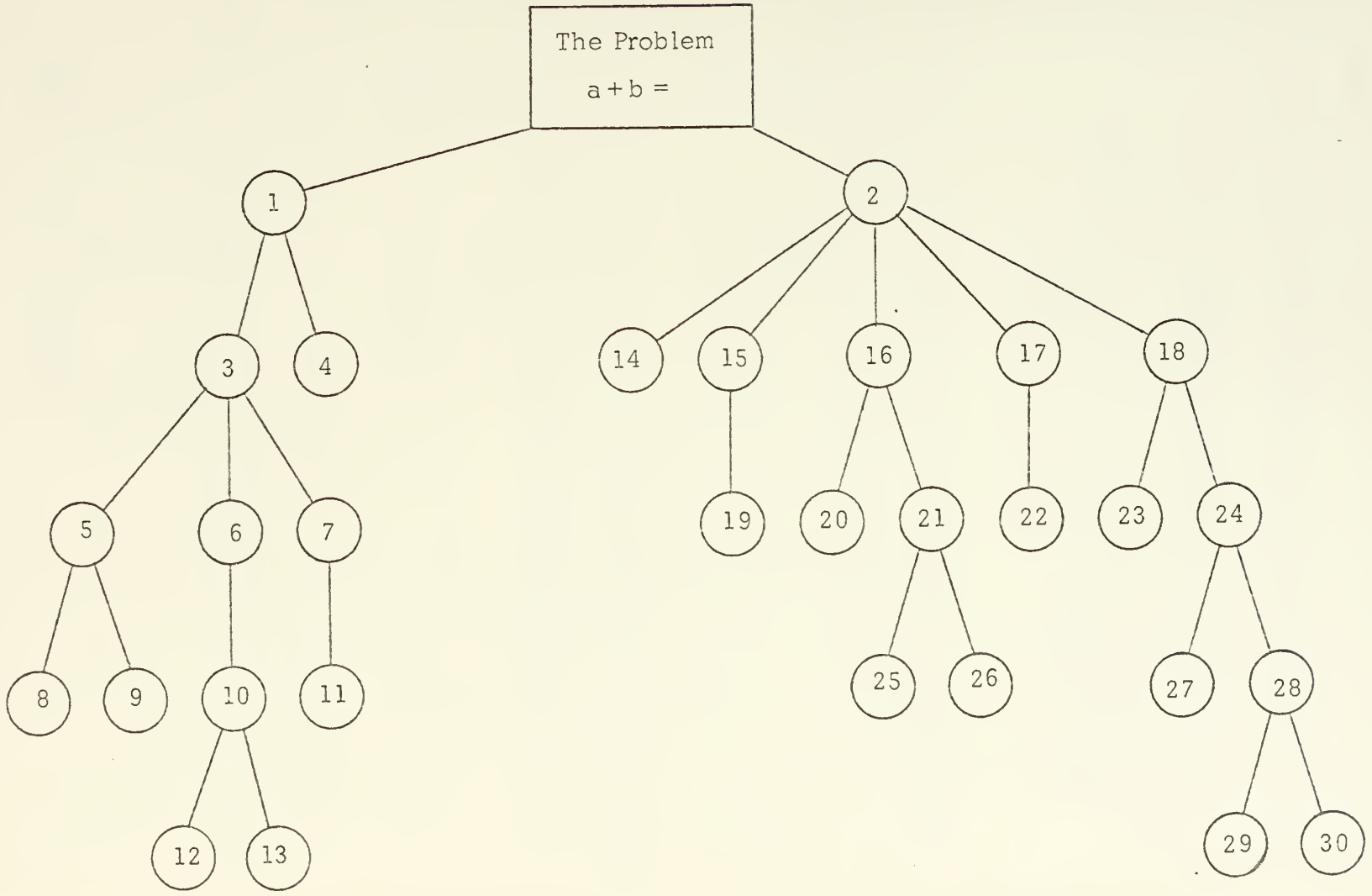
HARD: a=3, b=3, c=3, HU >10, TE >10, UN> 10

MEDIUM: a=2, b=2, c=2, TE >10, UN >10

EASY: a=2, b=2, c=1, TE >10, UN >10

SAMPLE TREE FOR LEVEL 1 PROBLEM TYPE 1

A. DIAGRAM OF TREE



B. DESCRIPTIONS OF CONDITIONS AND PROCESSES FOR SAMPLE TREE

The functions referred to in the nodes perform the following tasks:

1. Inverse - Changes a problem to the inverse of the problem, e.g., a+b=c c-a=b.
2. Number Line - Presents the student with a number line
3. Objects - Presents the student with a picture of the problem, e.g., 4+2= ; **** + **

NODE 1

The student has answered the question correctly, and the teacher desires to stress the communative property, so ask 'b+a' ;'.

COND: RIGHT;

PROC: PR: NEW, b+a= ;

NODE 2

The student has answered the question incorrectly, and the teacher desires to give him another chance before proceeding further.

COND: WRONG;

PROC: PR: REPEAT;

NODE 3

The student has answered the question 'b+a' correctly, and if it is a MEDIUM or EASY question, the teacher desires to drill him further on the family of facts by asking 'Answer-a= '.

COND: RIGHT & HARD;

PROC: FU: INVERSE;

NODE 4

The student has answered the question b+a incorrectly and the teacher desires to follow the same steps as for a+b.

COND: WRONG

PROC: GO: NODE 2;

NODE 5

The student has answered the question 'Answer-a= ' and for further drill, the teacher desires to ask 'Answer-b= '.

COND: RIGHT;

PROC: FU: INVERSE;

NODE 6

The student has answered the question 'Answer-b= ' incorrectly and appears to have added instead of subtracted.

COND: WRONG & SANS = a + b= ;

PROC: ST: You added when you should have subtracted.;

NODE 7

The student has answered the question 'Answer-a= ' incorrectly, so the teacher desires to give an example of the idea that subtraction is inverse of addition.

COND: WRONG;

PROC: ST: 7+3=10, 10-3=7;

NODE 8

The student has answered the question 'Answer-b=' correctly and the teacher desires to generate a new problem.

COND: RIGHT;

PROC: HA:;

NODE 9

The student has answered the question 'Answer-b= ' incorrectly, and the teacher desires to follow the same process cited when "Answer-a= ' is answered incorrectly.

COND: WRONG;

PROC: GO: NODE 3;

NODE 10

After presentation of the statement in Node 6, the teacher desires to give the student another chance at the problem.

COND: WRONG;

PROC: PR: REPEAT;

NODE 11

After presentation of the statement in the same node, the teacher wishes to follow the same process followed at Node 10.

COND: WRONG;

PROC: GO: NODE 10;

NODE 12

The student gets the question right after the appropriate hint, and is to get a chance at the problem 'Answer-b= '.

COND: RIGHT;

PROC: GO: NODE 5;

NODE 13

The student still cannot get the correct answer after the hint, and the teacher desires to give another problem.

COND: WRONG;

PROC: HALT;

NODE 14

The student answers the question correctly on the second try and is allowed to proceed.

COND: RIGHT;

PROC: GO: NODE 1;

NODE 15

The student gets the second try wrong and appears to be subtracting instead of adding.

COND: WRONG & SANS=a-b;

PROC: ST: You subtracted when you should have added.

NODE 16

The student gets the second try wrong and it appears he is not carrying. The teacher desires to show him an example.

COND: WRONG & HARD & MANS-SANS >=10;

PROC: PR: EXAMPLE;

NODE 17

The student gets the second try wrong and it appears he was just careless and the teacher warns him.

COND: WRONG & HARD & MANS-SANS <10;

PROC: ST: You added too hastily. Try it again and be more careful.

NODE 18

The student has gotten the second try wrong and none of the previous cases fit, so the teacher desires to use the number line to give him a clue.

COND: WRONG;

PROC: FU: NUMBER LINE;

NODE 19

Following the clue that the student is subtracting instead of adding, the teacher desires to repeat the question.

COND: WRONG;

PROC: GO: NODE 2;

NODE 20

After an example, the student gets the problem right, and the teacher desires that he continue.

COND: RIGHT;

PROC: GO: NODE 1;

NODE 21

Even with an example, the student still gets the question wrong, and so the teacher wants the lesson frame represented.

COND: WRONG;

PROC: GO: NODE 2;

NODE 22

After warning the student to be more careful, the teacher desires that the question be repeated.

COND: WRONG;

PROC: GO: NODE 2;

NODE 23

With the help of the number line, the student gets the question right and the teacher allows him to continue.

COND: RIGHT;

PROC: GO: NODE 1;

NODE 24

The number line fails to help the student so the teacher wants to give him some objects which he can count.

COND: WRONG;

PROC: FU: OBJECT;

NODE 25

After a representation of the lesson frame, the student gets the problem right and the teacher allows him to continue.

COND: RIGHT;

PROC: GO: NODE 1;

NODE 26

Even after the representation of the lesson frame the student cannot get the problem right, so the teacher decides to try another problem.

COND: WRONG;

PROC: HA:;

NODE 27

After being given objects to count, the student gets the correct answer, he is allowed to continue.

COND: RIGHT;

PROC: GO: NODE 1;

NODE 28

Even objects to count do not help the student, so the teacher decides to represent a previous lesson frame on the relations of numbers.

COND: WRONG;

PROC: FR: 0;

NODE 29

After the representation of the lesson frame, the student gets the correct answer, but is not allowed to continue and a new problem will be presented.

COND: RIGHT;

PROC: HA:;

NODE 30

Nothing helps the student, so the teacher decides to try another problem.

```
COND:    WRONG;
PROC:    HA:;
```

The conditions 'SANS=a-b' and 'SANS=a+b', and the process 'GO: NODE (number)' used in this example are not presently allowed in the language for specifying Trees. They were not included originally because other means of accomplishing the same ideas were available. They were intended to be included later in the development of RASCAL. In preparing the example, however, it was thought better to use these items for the sake of clarity. Further discussion of these items is located in Section V.

```
RASCAL:PROC OPTIONS(MAIN);

/*****************************************************/
/*

     RASCAL  IS THE MAIN PROGRAM AND PERFORMS THE
  SUPERVISORY FUNCTIONS OF THE SYSTEM. IT CALLS ON  THE
  ROUTINES  ESTAB, REFILE, PRO_GEN, PROB_SOL AND SHOW  TO
  PERFORM MAJOR TASKS. WITHIN THE BODY OF  RASCAL  ARE THE
  MECHANISMS TO OBTAIN NEEDED INFORMATION ABOUT THE STUDENT
  AND THE MECHANISM FOR DECIDING WHEN TO ADVANCE THE
  STUDENT. AS NOW WRITTEN, RASCAL  PERFORMS ONLY
  THE FUNCTIONS REQUIRED OF A DRILL AND PRACTICE SYSTEM.
  ALL VARIABLES OF MAJOR IMPORTANCE TO THE SYSTEM ARE DE -
  CLARED IN RASCAL. THEY PERFORM FUNCTIONS AS FOLLOWS:
      TREE - INPUT FILE OF LEVELS, PROBLEM TYPES AND TREES.
      COPTREE - OUTPUT FILE OF LEVELS, PROBLEM TYPES AND
                TREES.
      ALPHA- CHARACTER STRING USED TO TRANSLATE AN
             ARGUMENT INTO AN ENTRY POINT TO THE ARRAY
             ARG_PTRS.
      TRANS_D - ARRAY CONTAINING  KEY WORDS FOR RECOGNITION
                OF PLACE VALUE IN PROBLEM GENERATOR.
      COND_WORD - ARRAY CONTAINING  KEY WORDS  FOR
                  RECOGNITION BY THE INTERPRETER OF
                  CONDITIONS ON A NODES APPLICABILITY.
      PROB_S - STRING CONTAINING PROBLEM FORMAT.
      COND_S - STRING CONTAINING CONDITION LISTS - EITHER
               ON DIFFICULTY OF PROBLEM OR APPLICABILITY OF
               NODE.
      WORK_S - A COPY OF STRINGS WHICH WILL BE DESTROYED IN
               HANDLING, SO THE ORIGINALS WILL BE PRESERVED
      SHOW_STR - COPY OF THE PROBLEM;PASSED TO  SHOW  FOR
                 PRESENTATION TO THE STUDENT.
      HARD_L - THE LIST OF PROBLEMS MISSED BY THE STUDENT.
      HARD_LIST - PERMANENT COPY OF ALL PROBLEMS MISSED BY
                  THE STUDENT, TO BE USED AS REVIEW
                  QUESTIONS.
      BUFFER - USED TO READ RECORDS OR CARD DATA INTO
               STORAGE.
      HOLD - SAME AS BUFFER.
  *NOTE - REALLY ONLY TWO STRINGS ARE REQUIRED; HOWEVER,
          MORE ARE DECLARED TO GIVE THE NAMES MEANING.
      LEVELS - CONTAINS INFORMATION REGARDING A  LEVEL;
               LIMITED TO 10 FOR PRACTICALITY.
          NAME - NAME OF THE LEVEL.
          N_P - NUMBER OF PROBLEM TYPES IN THE LEVEL.
          N_S - NUMBER OF STUDENTS WHO HAVE ENTERED THE
                LEVEL.
          NR_P_A - NUMBER OF PROBLEMS ASKED IN THAT LEVEL,
                   DIVIDED BY N_S GIVES THE AVERAGE NUMBER
                   OF PROBLEMS REQUIRED BY A STUDENT WHICH
                   IS USED IN DETERMINING THE STUDENT'S
                   RATE.
          LEV_PTR - POINTER TO FIRST PROBLEM TYPE FOR THE
                    LEVEL.
          FRM_PTR - POINTER TO FRAME ASSOCIATED WITH LEVEL
                    WHEN IT IS IN CORE.
      SIZE - PARAMETER REQUIRED FOR ALLOCATION OF A VARIBLE
             AMOUNT OF BASED STORAGE.
      PROB_TYPE - CONTAINS THE ELEMENTS DEFINING A PROBLEM.
          N_O_S - NUMBER OF TIMES PROBLEM TYPE SELECTED
                  AND PRESENTED.
          PROB_I - POINTER TO INFIX EXPRESSION OF PROBLEM.
          PROB_P - POINTER TO POLISH EXPRESSION OF PROBLEM.
          HARD_P - POINTER TO LIST OF HARD CONDITIONS.
          MED_P - POINTER TO LIST OF MEDIUM CONDITIONS.
          EASY_P - POINTER TO LIST OF EASY CONDITIONS.
          ARG_P - POINTER TO LIST OF ARGUEMENTS USED IN
                  PROBLEM.
          P_NEXT - POINTER TO NEXT PROBLEM TYPE.
```

P_BRANCH - POINTER TO FIRST NODE OF TREE ASSOCIATED WITH PROBLEM TYPE.

PROBLEM_S - USED IN STORAGE OF BOTH THE POLISH AND INFIX EXPRESSIONS OF THE PROBLEM.

CONDITIONS - USED IN THE STORAGE OF ALL CONDITION STRINGS AND ALSO THE PROCESS TO BE CARRIED OUT AT A NODE.

NODE - A NODE IN A TREE.

COND_P - POINTER TO CONDITIONS OF APPLICABILITY.

PROC_P - POINTER TO PROCESS TO BE CARRIED OUT.

N_NEXT - POINTER TO NEXT NODE AT SAME LEVEL ASSOCIATED WITH THE SAME NODE AT A HIGHER LEVEL.

N_BRANCH - POINTER TO FIRST NODE AT THE NEXT LOWER LEVEL TO BE ASSOCIATED WITH THE NODE IN QUESTION.

RET_P - POINTER BACK TO THE MOTHER NODE.

SUCCESS - NUMBER OF TIMES A NODE IS ENTERED. BY COMPARING WITH SUCCESS OF NODE AT NEXT HIGHER LEVEL, A MEASURE OF THE WORTH OF THE NODE CAN BE OBTAINED.

PROBLEM - STORAGE FOR NUMBERS GENERATED BY PRO-GEN AND ALSO PARTIAL SOLUTIONS GENERATED BY PROBSOL, STORES EACH DIGIT OF NUMBER SEPARATELY.

ARGUEMENTS - STORAGE FOR THE ARGUMENT LIST.

PROB_STAT - CONTAINS INFORMATION ABOUT THE STATUS OF A PROBLEM TYPE.

DIF - LEVEL OF DIFFICULTY OF PROBLEMS TO BE GENERATED. TAKES ON FIVE POSSIBLE VALUES.

0 - FAILED AT EASY LEVEL.

1 - EASY PROBLEM.

2 - MEDIUM PROBLEM.

3 - HARD PROBLEM.

4 - PASSED PROB_TYPE.

SCORE - SCORE OF STUDENT ON THAT PROBLEM TYPE.

N_ASKED - NUMBER OF QUESTIONS ASKED.

TEMP_NODE - NODES IN THE SEQUECE OF PARTIAL ANSWERS.

T_P - POINTER TO NEXT TEMP_NODE.

T_A_P - POINTER TO LOCATION OF PARTIAL ANSWER.

PROB_S_P - ARRAY OF POINTERS POINTING TO PROB-STAT FOR CORRESPONDING PROBLEM TYPE.

ARG_PTRS - ARRAY OF POINTERS POINTING TO NUMBER TO REPLACE CORRESPONDING ARGUMENT. CONTROLLED SO IT CAN BE OVERLAYED BY A NEW PROBLEM IF NECESSARY.

CHOICE - ARRAY CONTAINING THE ALLOWED CHANGE IN THE ACTIVE PLACE VALUE DIGET OF THE ARGUEMENT CORRESPONDING TO THE ELEMENT OF CHOICE. FIVE VALUES MAY BE TAKEN.(THE DIGIT MUST REMAIN IN THE RANGE 0 - 9.)

0 - THE DIGIT MAY BE CHANGED INY MANNER.

1 - THE DIGIT MAY BE INCREASED ANY AMOUNT.

2 - THE DIGIT MAY BE DECREASED ANY AMOUNT.

3 - THE DIGIT MAY BE INCREASED OR DECREASED A MINIMAL AMOUT.

4 - THE DIGIT MAY NOT BE CHANGED.

REVUE - ARRAY WHICH KEEPS TRACK OF NUMBER OF TIMES PROBLEM TYPES MARKED FOR REVIEW ARE SELECTED AND ONLY ALLOWS PRESENTATION EVERY THIRD TIME

STAT_S - STRING CONTAINING STATUS FOR EACH PROBLEM TYPE IF LESSON TERMINATES PRIOR TO COMPLETING LEVEL.

M_ANS_C - THE COMPUTER'S ANSWER,EACH DIGIT STORED SEPARATELY AS A CHARACTER.

M_ANS - THE COMPUTER'S ANSWER STORED AS A NUMBER.

ANSWER - THE COMPUTER'S ANSWER STORED AS A CHARACTER STRING.

S_ANS_C - STUDENTS ANSWER AS SEPARATE CHARACTERS.

S_ANS - STUDENTS ANSWER AS A NUMBER.

ANS_CH - STUDENT'S ANSWER AS A CHARACTER STRING.

NEED - SEED FOR RANDOM NUMBER GENERATOR.

```
     S_RATE - THE STUDENTS ABILITY:
           0 - SLOW.
           1 - AVERAGE.
           2 - FAST.
     PROB_DIF - DIFFICULTY OF A PROBLEM. TAKES ON THE SAME
                VALUES AS DIF IN PROB_STAT.
     U_L - UPPER LIMIT ON RANDOM NUMBER.
     L_L - LOWER LIMIT ON RANDOM NUMBER.
     R_N - RANDOM NUMBER.
     ACTIVE - POINTER TO ACTIVE PROBLEM TYPE.
     N_ACTIVE - USED TO GET REQUIRED ELEMENTS FROM ACTIVE
                PROBLEM TYPE
  *NOTE - ACTIVE & N_ACTIVE ALSO USED IN ESTABLISHING AND
   REFILING TREES.
     P_PTR - POINTER TO ACTIVE 'PROBLEM'.
     ARG_PTR - POINTER TO ACTIVE ARGUMENT LIST.
     TEMP_PTR - POINTER TO HEAD OF LIST OF PARTIAL ANSWERS
     T_PTR - POINTER TO ELEMENTS OF PARTIAL ANSWER LIST.
     STAT_PTR - POINTER TO ACTIVE PROB_STAT
     TEST_PTR - POINTER USED IN DETERMINING IF STUDENT
  PASSED OR FAILED.
     R_FLAG - INDICATOR OF WHETHER PROBLEM IS A REVIEW
              PROBLEM.
     GR_FLAG - INDICATOR OF WHEN TO ASK QUESTIONS ON HARD
               LIST.
     RIGHT - NR. OF QUESTIONS ON  HARD_L ANSWERED
             CORRECTLY
     ASKED - NR OF QUESTIONS ON HARD_L ASKED.
     COMP_C - CORRECT ANSWER TO A QUESTION ON HARD_L AS A
              CHARACTER STRING
     COMP - CORRECT ANSWER TO QUESTION ON HARD_L AS A
            NUMBER
     ANS - STUDENT'S ANSWER TO QUESTION ON HARD_L.
     GRADE - STUDENT'S SCORE ON QUESTIONS FROM HARD_L.
     FAIL_COUNT - NR OF TIMES STUDENT FAILED TO ANSWER
                  SUFFICIENT QUESTIONS ON HARD_L TO PASS.
     COMMA, SEMI, EQUAL - POSITION OF NAMED SYMBOLS IN A
                          STRING.
     LEV_NR - NUMBER OF ACTIVE LEVEL
     LEV_C  - NUMBER OF ACTIVE LEVEL AS CHARACTER STRING.
     DEPTH_ANS - NR OF OVERLAYS OF ARG_PTRS.
     LONG - LENGTH OF VARYING CHARACTER STRING.

                                                          */
/*****************************************************************/


  DCL (TREE,COPTREE) FILE STREAM ENVIRONMENT(F(80));
  DCL PROBSOL ENTRY(CHAR(240) VARYING);
  DCL PRO_GEN ENTRY(CHAR(240) VARYING);
  DCL SHOW ENTRY(CHAR(240) VARYING,FIXED BIN(15));
  DCL RANDOM ENTRY(FIXED BIN(31),FIXED BIN(15),FIXED
                   BIN(15),FIXED BIN(15));
  DCL ALPHA CHAR(26) INITIAL('ABCDEFGHIJKLMNOPQRSTUVWXYZ');
  DCL  (COND_WORD(10) CHAR(2),TRANS_D(10) CHAR(2)) INITIAL
                                                   CALL INTL;
  DCL (PROB_S,COND_S) CHAR(240) VARYING;
  DCL WORK_S CHAR(240) VARYING;
  DCL SHOW_STR CHAR(240)VARYING;
  DCL HARD_L CHAR(240) VARYING;
  DCL HARD_LIST CHAR(240) VARYING;
  DCL BUFFER CHAR(80);
  DCL HOLD CHAR(80) VARYING;
  DCL 1 LEVELS(10),
        2 NAME CHAR(60),
        2 N_P FIXED BIN(15),
        2 N_S FIXED BIN(31),
        2 NR_P_A FIXED BIN(15),
        2 LEV_PTR PTR,
        2 FRM_PTR PTR;
  DCL SIZE FIXED BIN(15);
```

```
DCL 1 PROB_TYPE BASED(PTP),
       2 N_O_S FIXED BIN(31),
       2 PROB_I   PTR,
       2 PROB_P   PTR,
       2 HARD_P   PTR,
       2 MED_P    PTR,
       2 EASY_P   PTR,
       2 ARG_P    PTR,
       2 P_NEXT     PTR,
       2 P_BRANCH   PTR;
DCL 1 PROBLEM_S  BASED(PSP),
       2 P_NR FIXED BIN(15),
       2 PROB_STR CHAR(SIZE REFER(P_NR));
DCL 1 CONDITIONS BASED(CP),
       2 C_NR FIXED BIN(15),
       2 COND CHAR(SIZE REFER(C_NR));
DCL 1 NODE BASED(NP),
       2 COND_P PTR,
       2 PROC_P PTR,
       2 N_NEXT   PTR,
       2 N_BRANCH PTR,
       2 RET_P  PTR,
       2 SUCCESS FIXED BIN(15);
DCL 1  PROBLEM  BASED (PP),
       2 DIGETS FIXED BIN(15),
       2 PROB (SIZE REFER(DIGETS)) FIXED BIN(15);
DCL 1  ARGUEMENTS BASED(AP),
       2 NR FIXED BIN(15),
       2 ARGUE (SIZE REFER(NR)) CHAR(1);
DCL 1  PROB_STAT BASED(PSTP),
       2 DIF FIXED BIN(15),
       2 SCORE FLOAT BIN(21),
       2 N_ASKED FIXED BIN(15);
DCL 1 TEMP_NODE BASED(TNP),
       2 T_P PTR,
       2 T_A_P PTR;
DCL PROB_S_P(10) PTR;
DCL ARG_PTRS(26) CONTROLLED PTR;
DCL CHOICE(26) CONTROLLED FIXED BIN(15);
DCL REVUE(10) FIXED BIN(15);
DCL STAT_S CHAR(10) VARYING;
DCL M_ANS_C(10) CONTROLLED CHAR(1);
DCL M_ANS FIXED BIN(15);
DCL ANSWER CHAR(10) VARYING;
DCL S_ANS_C(10) CONTROLLED CHAR(1);
DCL S_ANS FIXED BIN(15);
DCL ANS_CH CHAR(10) VARYING;
DCL S_RATE FIXED BIN(15);
DCL PROB_DIF FIXED BIN(15);
DCL NEED FIXED BIN(31);
DCL (U_L,L_L) FIXED BIN(15);
DCL R_N FIXED BIN(15);
DCL (ACTIVE,N_ACTIVE,P_PTR) PTR;
DCL (ARG_PTR,TEMP_PTR,T_PTR) PTR;
DCL STAT_PTR PTR;
DCL TEST_PTR PTR;
DCL (GR_FLAG,R_FLAG) BIT(1);
DCL RIGHT FIXED BIN(15);
DCL ASKED FIXED BIN(15);
DCL COMP_C CHAR(10) VARYING;
DCL COMP FIXED BIN(15);
DCL ANS FIXED BIN(15);
DCL GRADE FLOAT BIN(21);
DCL FAIL_COUNT FIXED BIN(15);
DCL (COMMA,EQUAL) FIXED BIN(15);
DCL SEMI FIXED BIN(15);
DCL LEV_NR FIXED BIN(15);
DCL LEVEL_C CHAR(3) VARYING;
DCL DEPTH_ANS FIXED BIN(15);
DCL LONG FIXED BIN(15);
DCL NR_ST FIXED BIN(15);
DCL NR_ST_C CHAR(15);
```

```
/*****************************************************************/
/*
     ESTABLISH LEVELS,PROBLEM TYPES AND TREES IN CORE.
  ONCE THE LEVEL NUMBER IS KNOWN THE STATUS ARRAYS FOR THE
  NUMBER OF PROBLEMS AT THAT LEVEL ARE ALLOCATED.
  ALSO INITIALIZE RANDOM NR. GENERATOR.
                                                                */
/*****************************************************************/
  CALL ESTAB;
  DISPLAY('ENTER SEED'); GET LIST(NEED);
  CALL RANDOM(NEED,0,9,R_N);
/*****************************************************************/
/*
     GET ALL NECESSARY INFORMATION ABOUT A HYPOTHETICAL
  STUDENT. IN PRACTICE THIS INFORMATION WOULD BE OBTAINED
  FROM STUDENT FILES.
                                                                */
/*****************************************************************/
  HARD_L = '';
  DISPLAY('ENTER LEVEL NUMBER');
  GET LIST (LEV_NR);
  DO I = 1 TO N_P(LEV_NR);
     ALLOCATE PROB_STAT;
     PROB_S_P(I) = PSP;
  END;
  DO I = N_P(LEV_NR)+1 TO 10;
     PROB_S_P(I) = NULL;
  END;
MORE1:DISPLAY('ENTER HARD LIST')REPLY(HOLD);
/*****************************************************************/
/*
     IF THE HARD LIST IS BLANK THEN THE STUDENT IS
  STARTING IN THE LEVEL AFRESH. INITIALIZE PROB_STAT.
                                                                */
/*****************************************************************/
  IF HOLD = ' '
  THEN DO;
       N_S(LEVEL_NR) = N_S(LEV_NR) + 1;
       DO I = 1 TO 10 WHILE(PROB_S_P(I)¬= NULL);
            STAT_PTR = PROB_S_P(I);
            STAT_PTR -> DIF = 1;
            STAT_PTR -> SCORE = 0;
            STAT_PTR -> N_ASKED = 0;
       END;
  END;
  ELSE DO;
/*****************************************************************/
/*
     IF HARD LIST IS NOT BLANK THEN RESTABLISH THE
  PROB_STAT ARRAYS TO THE SAME VALUE THEY LAST HAD WHEN
  THE LESSON TERMINATED.
                                                                */
/*****************************************************************/
       SEMI = INDEX(HOLD,',');
       IF SEMI = 0
       THEN DO;
            HARD_L = HARD_L || HOLD;
            GO TO MORE1;
       END;
       HARD_L = HARD_L || SUBSTR(HOLD,1,SEMI-1);
       DISPLAY('ENTER STUDENT STATUS FOR EACH PROB_TYPE')
       REPLY(STAT_S);
       DO I = 1 TO 10 WHILE(SUBSTR(STAT_S,1,1) ¬= ' ');
            STAT_PTR = PROB_S_P(I);
            STAT_PTR -> DIF = SUBSTR(STAT_S,1,1);
            DISPLAY('ENTER SCORE');
            GET LIST (STAT_PTR->SCORE);
            DISPLAY('ENTER NUMBER ASKED');
            GET LIST (STAT_PTR->N_ASKED);
            STAT_S = SUBSTR(STAT_S,2);
       END;
  END;
```

```
/*************************************************************/
/*
      SET UP TO BEGIN ASKING QUESTIONS.
                                                             */
/*************************************************************/
  FAIL_COUNT = 0;
  ALLOCATE ARG_PTRS;
  ARG_PTRS = NULL;
  ALLOCATE CHOICE;
  CHOICE = 0;
/*************************************************************/
/*
      SELECT A PROBLEM TYPE TO ASK.
                                                             */
/*************************************************************/
R1:ACTIVE = LEV_PTR(LEV_NR);
  CALL RANDOM(0,1,N_P(LEV_NR),R_N);
  DO P_NR = 2 TO R_N;
      ACTIVE = ACTIVE -> P_NEXT;
  END;
  STAT_PTR = PROB_S_P(R_N);
/*************************************************************/
/*
    DETERMINE THE DIFFICULTY OF THE QUESTION TO BE ASKED
                                                             */
/*************************************************************/
  IF STAT_PTR -> DIF = 4 | STAT_PTR->DIF=0
  THEN GO TO REVIEW;
  IF STAT_PTR -> DIF = 1
  THEN DO;
      N_ACTIVE = ACTIVE -> EASY_P;
      WORK_S = N_ACTIVE -> COND;
 END;
  IF STAT_PTR -> DIF = 2
  THEN DO;
      N_ACTIVE = ACTIVE -> MED_P;
      WORK_S = N_ACTIVE -> COND;
  END;
  IF STAT_PTR -> DIF = 3
  THEN DO;
      N_ACTIVE = ACTIVE-> HARD_P;
      WORK_S = N_ACTIVE -> COND;
  END;
/*************************************************************/
/*
      GENERATE A QUESTION,CALCULATE AN ANSWER,DISPLAY THE
  QUESTION AND GET THE STUDENTS ANSWER.
                                                             */
/*************************************************************/
R3:ARG_PTR = ACTIVE -> ARG_P;
  CALL PRO_GEN(WORK_S);
  N_ACTIVE = ACTIVE -> PROB_I;
  SHOW_STR = N_ACTIVE -> PROB_STR;
  CALL SHOW(SHOW_STR,0);
  NR_P_A(LEV_NR) = NR_P_A(LEV_NR) + 1;
  N_ACTIVE = ACTIVE-> PROB_P;
  PROB_S = N_ACTIVE->PROB_STR;
  CALL PROBSOL(PROB_S);
  DISPLAY('ENTER ANSWER') REPLY(BUFFER);
  DO WHILE (SUBSTR(BUFFER,1,1)=' ');
      BUFFER = SUBSTR(BUFFER,2);
  END;
  ANS_CH = '';
  DO WHILE(SUBSTR(BUFFER,1,1) ¬= ' ');
      ANS_CH = ANS_CH||SUBSTR(BUFFER,1,1);
      BUFFER = SUBSTR(BUFFER,2);
  END;
  S_ANS = ANS_CH;
  LONG = LENGTH(ANS_CH);
  ALLOCATE S_ANS_C;
  S_ANS_C = ' ';
```

```
  DO I = LONG TO 1 BY -1;
       S_ANS_C(I) = SUBSTR(ANS_CH,1,1);
       ANS_CH = SUBSTR(ANS_CH,2);
  END;
  IF R_FLAG
/*****************************************************/
/*
       IF THIS WAS A REVIEW PROBLEM THEN ONLY CHECK TO SEE
  IF ANSWER OF STUDENT IS CORRECT, IF IT IS AND THE
  STUDENT IS FAILING THE PROBLEM TYPE THEN UPDATE THE SCORE
  IF STUDENTS ANSWER IS WRONG THEN ADD TO HARD_L AND GIVE
  CORRECT ANSWER. IF STUDENT IS FAILING PROB TYPE THEN
  CHECK TO SEE IF SCORE LESS THAN -1.0.
                                                      */
/*****************************************************/
  THEN DO;
       IF S_ANS ¬= M_ANS
       THEN DO;
            POSIT = INDEX(SHOW_STR,'?');
            SHOW_STR = SUBSTR(SHOW_STR,1,POSIT-1);
            IF HARD_L= ''
            THEN HARD_L = SHOW_STR || ANSWER;
            ELSE HARD_L = HARD_L||','||SHOW_STR || ANSWER;
            DISPLAY('THE CORRECT ANSWER IS  '||ANSWER||' .');
            GO TO NEXT_PROB;
       END;
       IF STAT_PTR->DIF = 0
       THEN DO;
            IF S_ANS = M_ANS
            THEN STAT_PTR->SCORE = STAT_PTR->SCORE+.2;
            ELSE STAT_PTR->SCORE = STAT_PTR->SCORE-.2;
            IF STAT_PTR->SCORE > -.5
                 THEN DO;
                 STAT_PTR-> DIF = STAT_PTR-> DIF + 1;
                 GO TO NEXT_PROB;
            END;
            ELSE GO TO CHFAIL;
       END;
  END;
/*****************************************************/
/*
       ADJUST THE STUDENT'S SCORE ON THE PROBLEM TYPE JUST
  ASKED.
                                                      */
/*****************************************************/
  IF S_ANS = M_ANS
  THEN DO;
       IF STAT_PTR -> DIF = 1
       THEN STAT_PTR -> SCORE = STAT_PTR -> SCORE + .2;
       ELSE STAT_PTR -> SCORE = STAT_PTR -> SCORE + .1;
  END;
  ELSE DO;
       POSIT  = INDEX(SHOW_STR,'?');
       SHOW_STR = SUBSTR(SHOW_STR,1,POSIT-1);
       IF HARD_L= ''
       THEN HARD_L = SHOW_STR || ANSWER;
       ELSE HARD_L = HARD_L||','||SHOW_STR || ANSWER;
       DISPLAY ('THE CORRECT ANSWER IS  '||ANSWER||' .');
       IF STAT_PTR -> DIF = 1
       THEN STAT_PTR -> SCORE = STAT_PTR-> SCORE - .2;
       ELSE STAT_PTR -> SCORE = STAT_PTR -> SCORE - .1;
  END;
  STAT_PTR -> N_ASKED = STAT_PTR-> N_ASKED + 1;
  IF STAT_PTR->SCORE >= .7
  THEN STAT_PTR->DIF = STAT_PTR->DIF + 1;
  IF STAT_PTR->SCORE <= -.5
  THEN STAT_PTR->DIF = STAT_PTR->DIF - 1;
  IF STAT_PTR ->DIF = 4
  THEN GO TO CHFIN;
  IF STAT_PTR->DIF = 0
  THEN GO TO CHFAIL;
```

```
 /*************************************************************/
 /*
      ERASE THE PARTIAL ANSWERS GENERATED IN SOLVING THE
   PROBLEM AND GENERATE THE NEXT PROBLEM.
                                                              */
 /*************************************************************/
NEXT_PROB:DO I = 1 TO TEMP;
      T_PTR = TEMP_PTR-> T_P;
      FREE TEMP_PTR->TEMP_NODE;
      TEMP_PTR = T_PTR;
   END;
   DO I = 1 TO 26;
      IF ARG_PTRS(I) ¬= NULL
      THEN DO;
           T_PTR = ARG_PTRS(I);
           FREE T_PTR->PROBLEM;
           ARG_PTRS(I) = NULL;
      END;
   GO TO R1;
 /*************************************************************/
 /*
      CHECK IF ALL PROBLEM TYPES IN THE LEVEL HAVE
   BEEN SUCCESSFULLY COMPLETED. IF SO THEN ASK QUESTIONS ON
   HARD_L AND DETERMINE GRADE FOR STUDENT. IF GRADE > .70
   THEN TERMINATE THE LESSON. ELSE RESET PROBLEM DIFFICULTY
   IN ALL PROBLEM TYPES AND TRY AGAIN.
                                                              */
 /*************************************************************/
CHFIN:DO I=1 TO 10 WHILE(PROB_S_P(I)¬=NULL);
      TEST_PTR = PROB_S_P(I);
      IF TEST_PTR->DIF < 4
      THEN GO TO R1;
   END;
   RIGHT = 0; ASKED = 0;
   IF HARD_L = ''
   THEN DO;
      LEV_NR = LEV_NR + 1;
      GO TO FIN;
   END;
   GR_FLAG = '0'B;
   WORK_S = '';
R2:COMMA = INDEX(HARD_L,',');
   IF COMMA = 0
   THEN DO;
      HOLD = HARD_L;
      GR_FLAG = '1'B;
   END;
   ELSE DO;
      HOLD = SUBSTR(HARD_L,1,COMMA-1);
      HARD_L = SUBSTR(HARD_L,COMMA+1);
   END;
   EQUAL = INDEX(HOLD,'=');
   COMP_C = SUBSTR(HOLD,EQUAL+1);
   COMP = COMP_C;
   HOLD = SUBSTR(HOLD,1,EQUAL);
   ASKED = ASKED + 1;
   DISPLAY(' ENTER ANSWER');
   GET LIST (ANS);
   IF COMP = ANS
   THEN DO;
      RIGHT = RIGHT + 1;
      IF GR_FLAG THEN GO TO GRAD;
      ELSE GO TO R2;
   END;
   ELSE DO;
      WORK_S = WORK_S||HOLD||COMP_C||',';
      IF GR_FLAG
      THEN GO TO GRAD;
      ELSE GO TO R2;
   END;
```

```
GRAD:IF WORK_S ¬=''
   THEN DO;
        LONG = LENGTH(WORK_S);
        WORK_S = SUBSTR(WORK_S,1,LONG-1);
        HARD_L = WORK_S;
   END;
   GRADE = RIGHT/ASKED;
   IF GRADE > .70
   THEN DO;
        HARD_LIST = HARD_LIST||HARD_L ||';';
        LEV_NR = LEV_NR + 1;
        GO TO FIN;
   END;
   ELSE DO;
        FAIL_COUNT = FAIL_COUNT +1;
        IF FAIL_COUNT > 3
        THEN GO TO TEACH1;
        DO I = 1 TO 10 WHILE (PROB_S_P(I)¬=NULL);
             STAT_PTR = PROB_S_P(I);
             STAT_PTR -> DIF = STAT_PTR->DIF - FAIL_COUNT;
             GO TO NEXT_PROB;
        END;
   END;
 /*************************************************************/
 /*
        IF THE SCORE FOR A PROBLEM TYPE GOES BELOW -1.0 OR
   A GIVEN PERCETAGE OF PROBLEM TYPES HAVE SCORES BELOW -.5
   THEN DEMOTE THE STUDENT AND NOTIFY THE TEACHER.
                                                             */
 /*************************************************************/
CHFAIL: IF STAT_PTR ->SCORE < -1
   THEN DO;
        LEV_NR = LEV_NR -1;
        GO TO TEACH2;
   END;
   K = 0;
   DO I = 1 TO 10 WHILE(PROB_S_P(I)¬=NULL);
        TEST_PTR = PROB_S_P(I);
        IF TEST_PTR -> DIF = 0
        THEN COUNT = COUNT +1;
        K = K + 1;
   END;
   FAIL = COUNT/K;
   IF S_RATE = 0
   THEN IF FAIL > .25
        THEN DO;
             LEV_NR = LEV_NR -1;
             GO TO TEACH2;
        END;
   IF S_RATE = 1
   THEN IF FAIL > .50
        THEN DO;
             LEV_NR = LEV_NR - 1;
             GO TO TEACH2;
        END;
   IF S_RATE = 2
   THEN IF FAIL > .75
        THEN DO;
             LEV_NR = LEV_NR -1;
             GO TO TEACH2;
        END;
   GO TO NEXT_PROB;
 /*************************************************************/
 /*
        CHECK TO SEE IF THIS IS THE THIRD TIME A PROBLEM
   MARKED FOR REVIEW HAS BEEN SELECTED.IF IT IS ALLOW THE
   PROBLEM TO BE ASKED. ELSE GO SELECT ANOTHER PROBLEM TYPE
                                                             */
 /*************************************************************/
```

```
REVIEW:IF REVUE(R_N) = 3
   THEN DO;
        R_FLAG = '1'B;
        REVUE(R_N) = 0;
        IF STAT_PTR -> DIF = 4
        THEN DO;
             N_ACTIVE = ACTIVE -> HARD_P;
             WORK_S = N_ACTIVE -> COND;
        END;
        ELSE DO;
             N_ACTIVE = ACTIVE -> EASY_P;
             WORK_S = N_ACTIVE -> COND;
        END;
        GO TO R3; END;
   ELSE DO;
        REVUE(R_N) = REVUE(R_N) + 1;
        GO TO R1;
   END;
TEACH1: DISPLAY('STUDENT HAS FAILED TO ADVANCE IN 4 TRIES.')
         DISPLAY(' ASSISTANCE REQUIRED');
         GO TO FIN;
TEACH2:DISPLAY('STUDENT LEVEL REDUCED BY ONE.');
         GO TO FIN;
FIN:DISPLAY('              LESSON COMPLETED.                  ')
 /*************************************************************/
 /*
        RESTORE THE TREES,PROBLEM TYPES AND LEVELS TO BACK-UP
   STORAGE.
                                                              */
 /*************************************************************/
   CALL REFILE;
END RASCAL;
```

```
INTL: PROC;

/*****************************************************/
/*
     INTL INITIALIZES ALL THE KEY WORD ARRAYS USED IN
  RASCAL.
                                                     */
/*****************************************************/

   TRANS_D(1) ='UN';
   TRANS_D(2) ='TE';
   TRANS_D(3) ='HU';
   TRANS_D(4) ='TH';
   TRANS_D(5) ='TT';
   TRANS_D(6) ='HT';
   TRANS_D(7) ='MI';
   TRANS_D(8) ='TM';
   TRANS_D(9) ='HM';
   TRANS_D(10) = 'BI';
   COND_WORD(1) = 'RI';
   COND_WORD(2) = 'WR';
   COND_WORD(3) = 'FA';
   COND_WORD(4) = 'AV';
   COND_WORD(5) = 'SL';
   COND_WORD(6) = 'HA';
   COND_WORD(7) = 'ME';
   COND_WORD(8) = 'EA';
   COND_WORD(9) = 'SA';
   COND_WORD(10) = 'MA';
END INTL;


        RANDOM: PROC(TYPE,L_L,U_L,RN);

/*****************************************************/
/*
     RANDOM RETURNS A PSUEDO-RANDOM NUMBER IN THE INTERVAL
  SET BY U_L AND L_L. IF TYPE IS ZERO THEN THE NEXT VALUE
  IN THE SEQUENCE IS TAKEN. IF TYPE IS A POSITIVE INTEGER
  THEN THE SEQUENCE STARTS AT THE VALUE DETERMINED BY TYPE.
  IF TYPE IS NEGATIVE THEN THE SEQUENCE STARTS AGAIN.
                                                     */
/*****************************************************/

   DCL(TYPE,RV STATIC INIT(3587)) FIXED BIN(31);
   DCL RL FIXED BIN(31);
   DCL R FLOAT BIN(21) STATIC;
   DCL RD FLOAT BIN(21);
   DCL(U_L,L_L) FIXED BIN(15);
   DCL RN FIXED BIN(15);
   IF  TYPE ¬= 0
   THEN IF TYPE < 0
        THEN RV = 3587;
        ELSE RV = TYPE;
   RV = MOD(RV * 3587,524288);
   R = RV;
   R = MOD(R,32768)/32768;
   RD =((R *(U_L - L_L +1)) + L_L);
   RL = RD;
   RN = RL;
END RANDOM;
```

```
                    SHOW:PROC(SHOW_S,N);

 /*****************************************************************/
 /*
       SHOW DISPLAYS THE PROBLEM TO THE STUDENT, REPLACING
   THE ARGUEMENTS IN THE PROBLEM FORMAT WITH THE NUMBERS
   GENERATED BY PRO_GEN. ALSO '*' ARE CHANGED TO 'X' TO MAKE
   THE PROBLEM MORE LIKE WHAT THE STUDENT IS EXPECTING.
       SHOW_S -  A COPY OF THE PROBLEM FORMAT.
       ARG_N  - THE NUMBER IN THE ARRAY ARG_PTRS WHERE THE
                POINTER,TO THE NUMBER WHICH IS TO REPLACE AN
                ARGUMENT, IS STORED.
                                                              */
 /*****************************************************************/

   DCL SHOW_S CHAR(240)VARYING; DCL P_PTR PTR;
   DCL N FIXED BIN(15);         DCL NUMBER_C CHAR(9)VARYING;
   DCL ARG_N FIXED BIN(15);     DCL ARG CHAR(1);
   DCL NUM FIXED BIN(15);
S1:IF  N= 0
   THEN DO;
 /*****************************************************************/
 /*
       FOR EACH NON NULL POINTER IN ARG_PTRS SEARCH SHOW_S
   FOR THE OCCURANCE OF THE ARGUMENT CORRESPONDING TO ARG_N
   AND REPLACE IT.
                                                              */
 /*****************************************************************/
    S2:DO ARG_N = 1 TO ARG_PTR->NR;
         ARG = ARG_PTR->ARGUE(ARG_N);
         NUM = INDEX(ALPHA,ARG);
         P_PTR = ARG_PTRS(NUM);
         NUMBER = 0;
         DO I = 1 TO P_PTR->DIGETS;
            NUMBER = NUMBER +(P_PTR->PROB(I)*(10**(I-1)));
         END;
       S3:POSIT  = INDEX(SHOW_S,ARG);
          NUMBER_C = NUMBER;
          DO WHILE(SUBSTR(NUMBER_C,1,1) =' ');
             NUMBER_C = SUBSTR(NUMBER_C,2);
          END;
          IF POSIT = 0
          THEN GO TO SOUT1;
          C_SHOW_S = '';
          IF POSIT ¬= 1
          THEN C_SHOW_S = SUBSTR(SHOW_S,1,POSIT -1)||' ';
          SHOW_S = NUMBER_C||' '||SUBSTR(SHOW_S,POSIT+1);
          SHOW_S = C_SHOW_S||SHOW_S;
          GO TO S3;
 SOUT1:END;
 /*****************************************************************/
 /*
       REPLACE '*' WITH 'X'.
                                                              */
 /*****************************************************************/
    S4:POSIT = INDEX(SHOW_S,'*');
       IF POSIT ¬= 0
       THEN DO;
            SHOW_S = SUBSTR(SHOW_S,1,POSIT-1)||'X'||
                                SUBSTR(SHOW_S,POSIT+1);
            GO TO S4;
       END;
       POSIT = INDEX(SHOW_S,';');
       SHOW_S = SUBSTR(SHOW_S,1,POSIT-1)||'?';
       DISPLAY(SHOW_S);
       RETURN;
   END;
END SHOW;
```

```
PRO_GEN: PROC(WORK_S);

/*************************************************************/
/*

     PRO_GEN HANDLES THE GENERATION OF NUMBERS TO REPLACE
  THE ARGUMENTS IN THE COPY OF THE PROBLEM FORMAT WHICH IS
  PASSED AS A PARAMETER. THE NUMBERS ARE STORED IN THE
  BASED ARRAY PROBLEM. POINTERS IN ARG_PTRS ARE ESTABLISHED
  TO INDICATE WHERE THE NUMBER IS STORED AFTER GENERATION.
  KEY ELEMENTS USED ONLY IN PRO_GEN ARE:
     WORK_S - CONTAINS THE COPY OF THE PROBLEM FORMAT.
     HOLD - CONTAINS THE CONDITION BEING OPERATED ON.
     GT - INDICATES WHERE IN WORK_S THE OPERATOR '>' IS
          FOUND.
     LT - INDICATES WHERE IN WORK_S THE OPERATOR '<' IS
          FOUND.
     EQ - INDICATES WHERE IN WORK_S THE OPERATOR '=' IS
          FOUND.
     LE - INDICATES WHERE IN WORK_S THE OPERATOR '<=' IS
          FOUND.
     GE - INDICATES WHERE IN WORK_S THE OPERATOR '>=' IS
          FOUND.
     DI - INDICATES WHERE IN WORK_S THE OPERATOR '//' IS
          FOUND.
     MU - INDICATES WHERE IN WORK_S THE OPERATOR '**' IS
          FOUND.
     COMMA - INDICATES WHERE IN WORK_S THE OPERATOR ',' IS
             FOUND.
     SEMI - INDICATES WHERE IN WORK_S THE OPERATOR ';' IS
            FOUND.
     DIFFER - CONTAINS THE DIFFERENCE BETWEEN THE ACTUAL
              NUMBER AND THE NUMBER REQUIRED TO SATISFY THE
              CONDITION BEING OPERATED ON.
     COMP - NUMBER WHICH IS REQUIRED BY THE CONDITION.
     REMAINDER - AMOUNT BY WHICH THE NUMBER BEING EXAMINED
                 MUST BE CHANGED TO SATISFY THE '//' AND
                 '**' OPERATORS.
     PLACE_C - CONTAINS THE CHARACTERS WHICH REPRESENT THE
               PLACE VALUE ON WHICH CONDITIONS ARE BEING
               SEARCHED FOR.

                                                          */
/*************************************************************/

  DCL DIV_FLAG BIT(1);
  DCL TOTAL FIXED BIN(15);
  DCL ANS CHAR(1);
  DCL LAST FIXED BIN(15);
  DCL (CARG1,CARG2) CHAR(1);
  DCL WORK_S  CHAR(240) VARYING;
  DCL C_WORK_S CHAR(240) VARYING;
  DCL HOLD CHAR(15) VARYING;
  DCL (POSIT,NUM,OP_TOT,LIMIT) FIXED BIN(15);
  DCL ARG CHAR(1);
  DCL G_DIG FIXED BIN(15) INITIAL(0);
  DCL PLACE_C CHAR(2);
  DCL (GT,LT,EQ,GE,LE,DI,MU) FIXED BIN(15);
  DCL LAB_L(0:9) LABEL;
  DCL LAB(8) LABEL;
  DCL OPTR CHAR(2) VARYING;
  DCL (ARG1,ARG2,NUM1,NUM2) FIXED BIN(15);
  DCL (P1,P2) PTR;
  DCL (DIFFER,COMP,CARRY,LAB_NR,REMAIN) FIXED BIN(15);
  DCL FLAG BIT(1);
  DCL P_PTR PTR;
  DCL (COMMA,SEMI) FIXED BIN(15);
```

```
/*****************************************************************/
/*
      THIS SEGMENT OF CODING DETERMINES WHICH ARGUMENTS ARE
  IN THE PROBLEM FORMAT, DETERMINES THE SIZE OF THE NUMBERS
  REQUIRED TO REPLACE THE ARGUMENTS AND ALLOCATES THE
  STORAGE FOR NUMBERS.
                                                              */
/*****************************************************************/
  DO  I = 1 TO  ARG_PTR-> NR;
      ARG  = ARG_PTR -> ARGUE(I);
      NUM = INDEX(ALPHA,ARG);
      POSIT = INDEX(WORK_S, ARG||'=');
      C_WORK_S = '';
      IF POSIT =1
      THEN COMMA = INDEX(WORK_S,',');
      ELSE DO;
           C_WORK_S = SUBSTR(WORK_S,1,POSIT-1);
           WORK_S = SUBSTR(WORK_S,POSIT);
           COMMA = INDEX(WORK_S,',');
      END;
      IF COMMA = 0
      THEN DO;
           SEMI = INDEX(WORK_S,';');
           HOLD = SUBSTR(WORK_S,1,SEMI-1);
           WORK_S = '';
           IF C_WORK_S ¬= ''
           THEN C_WORK_S=SUBSTR(C_WORK_S,1,LENGTH(C_WORK_S)
                                                  -1)||';' ;
      END;
      ELSE DO;
           HOLD = SUBSTR(WORK_S,1,COMMA-1);
           WORK_S = SUBSTR(WORK_S,COMMA+1);
      END;
      SIZE = SUBSTR(HOLD,3);
      WORK_S = C_WORK_S || WORK_S;
      ALLOCATE PROBLEM;
      ARG_PTRS(NUM)  = PP;
      IF  SIZE > G_DIG
      THEN G_DIG = SIZE;
  END;
/*****************************************************************/
/*
      THIS DO LOOP WHICH ENDS AT THE LABEL OUT1 CONTAINS
  THE MECHANISMS FOR SATISFYING THE CONDITIONS BELONGING TO
  GROUP 1 AND GROUP 2.
                                                              */
/*****************************************************************/
N1:DO I = 1 TO G_DIG;
      PLACE_C = TRANS_D(I);
      IF WORK_S = ''
      THEN DO;
           LAST = I;
           GO TO FILL;
      END;
/*****************************************************************/
/*
      THIS DO LOOP WHICH ENDS AT THE LABEL OUT:LAB(8)
  CONTAINS THE MECHANISMS FOR SATISFYING CONDITIONS
  BELONGING TO GROUP 1.
                                                              */
/*****************************************************************/
   N2:DO J = 1 TO ARG_PTR->NR;
           ARG = ARG_PTR -> ARGUE(J);
           NUM = INDEX(ALPHA,ARG);
           P_PTR = ARG_PTRS(NUM);
           CHOICE(NUM) = 0;
           U_L = 9;
           IF P_PTR->DIGETS = I & I>1
           THEN L_L = 1;
           ELSE L_L = 0;
           IF P_PTR -> DIGETS < I
           THEN GO TO OUT;
```

```
/*****************************************************/
/*
     THE FIRST STEP IS TO FIND ALL CONDITIONS IN THE FORM
  'PLACE VALUE (ARG) OPERATOR'.
                                                      */
/*****************************************************/
          GT = INDEX(WORK_S,PLACE_C||'('||ARG||')>');
          LT = INDEX(WORK_S,PLACE_C||'('||ARG||')<');
          EQ = INDEX(WORK_S,PLACE_C||'('||ARG||')=');
          GE = INDEX(WORK_S,PLACE_C||'('||ARG||')>=');
          LE = INDEX(WORK_S,PLACE_C||'('||ARG||')<=');
          DI = INDEX(WORK_S,PLACE_C||'('||ARG||')//');
          MU = INDEX(WORK_S,PLACE_C||'('||ARG||')**');
/*****************************************************/
/*
     NOW WE DETERMINE WHICH OPERATORS HAVE BEEN FOUND BY
  ASSIGNING A UNIQUE NUMBER TO EACH OPERATOR. THE TOTAL OF
  THESE NUMBERS THEN INDICATES THE OPERATORS FOUND.
                                                      */
/*****************************************************/
          OP_TOT = 0;
          IF LT ¬= 0
          THEN OP_TOT = OP_TOT + 1;
          IF GT ¬= 0
          THEN OP_TOT = OP_TOT + 2;
          IF EQ ¬= 0
          THEN OP_TOT = OP_TOT + 6;
          IF LE ¬= 0
          THEN OP_TOT = OP_TOT + 40;
          IF GE ¬= 0
          THEN OP_TOT = OP_TOT + 50;
          IF DI ¬= 0
          THEN OP_TOT = OP_TOT + 700;
          IF MU ¬= 0
          THEN OP_TOT = OP_TOT + 800;
          DIV_FLAG ='0'B;
          IF OP_TOT = 0
          THEN GO TO LAB_L(0);
/*****************************************************/
/*
     THE NEXT SECTION OF CODE DETERMINES THE OPERATORS
  WHICH WERE FOUND AND THEN BRANCHES TO THE LABEL WHICH
  CONTAINS THE MECHANISM FOR SATISFYING THAT OPERATOR.
                                                      */
/*****************************************************/
          NUM = OP_TOT/100;
     LAB0:IF NUM > 0
          THEN GO TO LAB_L(NUM);
     LAB1:NUM = OP_TOT/10;
          IF NUM > 0
          THEN DO;
               NUM = MOD(OP_TOT,100)/10;
               IF NUM > 0
               THEN GO TO LAB_L(NUM);
          END;
     LAB2:NUM = MOD (OP_TOT,10);
          GO TO LAB_L(NUM);
     LAB3:LAB_L(0):
          CALL RANDOM(0,L_L,U_L,R_N);
          GO TO INSERT;
 LAB_L(1): /* LESS THEN */
          IF  SUBSTR(WORK_S,LT+6,1)<'1'
          THEN DO;
                    LT = 0;
                    GO TO LAB3;
               END;
          LIMIT = SUBSTR(WORK_S,LT+6,1);
          U_L = LIMIT -1;
          CHOICE(NUM) = CHOICE(NUM) + 2;
          GO TO LAB3;
 LAB_L(2):/* GREATER THAN */
          IF SUBSTR(WORK_S,GT+6,1) < '1'
```

```
          THEN DO;
                    GT = 0;
                    GO TO LAB3;
               END;
          LIMIT = SUBSTR(WORK_S,GT+6,1);
          L_L = LIMIT + 1;
          CHOICE(NUM) = CHOICE(NUM)+1;
          GO TO LAB3;
LAB_L(3): /*GREATER THAN AND LESS THAN */
          IF SUBSTR(WORK_S,LT+6,1) < '1'
          THEN DO;
                    LT = 0;
                    GO TO LAB_L(2);
               END;
          LIMIT = SUBSTR(WORK_S,LT+6,1);
          U_L = LIMIT - 1;
          CHOICE(NUM) = CHOICE(NUM) + 2;
          GO TO LAB_L(2);
LAB_L(4): /* LESS THAN OR EQUAL */
          IF SUBSTR(WORK_S,LE+7,1) < '1'
          THEN DO;
                    LE = 0;
                    GO TO LAB2;
               END;
          U_L = SUBSTR(WORK_S,LE+7,1);
          IF LE = LT
          THEN DO;
               OP_TOT = OP_TOT - 1;
               LT = 0;
          END;
          CHOICE(NUM) = CHOICE(NUM) + 2;
          GO TO LAB2;
LAB_L(5): /* GREATER THAN OR EQUAL */
          IF SUBSTR(WORK_S,GE+7,1) < '1'
          THEN DO;
                    GE = 0;
                    GO TO LAB2;
               END;
          L_L = SUBSTR(WORK_S,GE+7,1);
          IF GT = GE
          THEN DO;
               OP_TOT = OP_TOT - 2;
               GT = 0;
          END;
          CHOICE(NUM) = CHOICE(NUM) +1;
          GO TO LAB2;
LAB_L(6): /*  EQUAL */
          IF SUBSTR(WORK_S,EQ+6,1) < '1'
          THEN DO;
                    EQ = 0;
                    GO TO INSERT;
               END;
          R_N = SUBSTR(WORK_S,EQ+6,1);
          CHOICE(NUM) = CHOICE(NUM)+4;
          GO TO INSERT;
LAB_L(7):LAB_L(8):
          DIV_FLAG = '1'B;
          GO TO LAB1;
LAB_L(9): /* LESS THAN OR EQUAL AND GREATER THAN OR EQUAL
          IF SUBSTR(WORK_S,LE+7,1)<'1'
          THEN DO;
                    LE = 0;
                    GO TO LAB_L(5);
               END;
          IF LT = LE
          THEN DO;
               LT = 0;
               OP_TOT = OP_TOT -1;
          END;
          U_L = SUBSTR(WORK_S,LE+7,1);
          CHOICE(NUM) = CHOICE(NUM) + 2;
          GO TO LAB_L(5);
```

```
/*****************************************************/
/*
     AFTER THE NUMBER HAS BEEN GENERATED IT IS STORED IN
  THE PROPER LOCATION BY THE NEXT SEGMENT OF CODING.
                                                      */
/*****************************************************/
  INSERT:
          IF DIV_FLAG
          THEN DO;
                  IF DI ¬= 0
                  THEN  LIMIT = SUBSTR(WORK_S,DI+7,1);
                  IF MU ¬= 0
                  THEN  LIMIT = SUBSTR(WORK_S,MU+7,1);
                  REMAIN = MOD(R_N,LIMIT);
                  IF  REMAIN ¬=0
                  THEN DO;
                          IF CHOICE(NUM)=1&R_N+LIMIT-REMAIN<
                          THEN DO;
                                  R_N=R_N+(LIMIT-REMAIN);
                                  GO TO PUT;
                               END;
                          IF CHOICE(NUM)= 2 & R_N-REMAIN>=0
                          THEN DO;
                                  R_N = R_N - REMAIN;
                                  GO TO PUT;
                               END;
                           IF R_N - REMAIN >= 0
                           THEN DO;
                                  R_N = R_N - REMAIN;
                                  GO TO PUT;
                               END;
                           IF R_N +(LIMIT - REMAIN) < 10
                           THEN DO;
                                  R_N = R_N+(LIMIT-REMAIN);
                                  GO TO PUT;
                               END;
                       END;
                  CHOICE(NUM) = 4;
              END;
     PUT:P_PTR -> PROB(I) = R_N;
/*****************************************************/
/*
     WHEN ALL CONDITIONS, BELONGING TO GROUP 1, ON A
  PARTICULAR PLACE VALUE AND ARGUMENT HAVE BEEN SATISFIED,
  THE CONDITIONS ARE REMOVED FROM WORK_S BY THE NEXT SEG-
  MENT OF CODE. THE PROCESS THEN RETURNS TO THE DO LOOP
  BEGINNING AT THE LABEL N2 TO FIND CONDITIONS, IN GROUP 1,
  ON THE SAME PLACE VALUE AND THE NEXT ARGUMENT.
                                                      */
/*****************************************************/
          LAB_NR = 1;
   LAB(1):IF GT > 0
          THEN DO;
                  POSIT = GT;
                  GO TO REMOVE;
               END;
          LAB_NR = LAB_NR + 1;
   LAB(2):IF LT > 0
          THEN DO;
                  POSIT = LT;
                  GO TO REMOVE;
               END;
          LAB_NR = LAB_NR + 1;
   LAB(3):IF EQ > 0
          THEN DO;
                  POSIT = EQ;
                  GO TO REMOVE;
               END;
          LAB_NR = LAB_NR + 1;
   LAB(4):IF LE> 0
          THEN DO;
                  POSIT = LE;
```

```
                    GO TO REMOVE;
               END;
          LAB_NR = LAB_NR + 1;
LAB(5):   IF GE > 0
          THEN DO;
                    POSIT = GE;
                    GO TO REMOVE;
               END;
          LAB_NR = LAB_NR+ 1;
LAB(6):   IF DI > 0
          THEN DO;
                    POSIT = DI;
                    GO TO REMOVE;
               END;
          LAB_NR = LAB_NR + 1;
LAB(7):   IF MU > 0
          THEN DO;
                    POSIT = MU;
                    GO TO REMOVE;
               END;
          GO TO OUT;
REMOVE:
          C_WORK_S = '';
          IF POSIT = 1
          THEN COMMA = INDEX(WORK_S,',');
          ELSE DO;
                    C_WORK_S = SUBSTR(WORK_S,1,POSIT - 1);
                    WORK_S = SUBSTR(WORK_S,POSIT);
                    COMMA = INDEX(WORK_S,',');
               END;
          IF COMMA = 0
          THEN DO;
               SEMI = INDEX(WORK_S,';');
               HOLD = SUBSTR(WORK_S,1,SEMI);
               WORK_S = '';
               IF C_WORK_S ¬= ''
               THEN C_WORK_S = SUBSTR(C_WORK_S,1,LENGTH(
                                        C_WORK_S)-1)||';' ;
          END;
          ELSE DO;
               HOLD = SUBSTR(WORK_S,1,COMMA);
               WORK_S = SUBSTR(WORK_S,COMMA+1);
          END;
          LONG = LENGTH(HOLD);
          WORK_S = C_WORK_S || WORK_S;
          IF GT > POSIT
          THEN GT = GT - LONG;
          IF LT > POSIT
          THEN LT = LT - LONG;
          IF EQ > POSIT
          THEN EQ = EQ - LONG;
          IF LE > POSIT
          THEN LE = LE - LONG;
          IF GE > POSIT
          THEN GE = GE - LONG;
          IF DI > POSIT
          THEN DI = DI - LONG;
          IF MU > POSIT
          THEN MU = MU - LONG;
          WORK_S = C_WORK_S ||WORK_S;
          LAB_NR = LAB_NR + 1;
          GO TO LAB(LAB_NR);
/*****************************************************/
/*
     WHEN ALL CONDITIONS, BELONGING TO GROUP 1, HAVE BEEN
  SATISFIED FOR ALL ARGUMENTS, CONTROL PASSES OUT OF THE
  LOOP BEGINNING AT THE LABEL N2 INTO THE NEXT SEGMENT OF
  CODING WHICH GETS THE NEXT CONDITION ON THE PLACE VALUE
  BEING HANDLED AND TESTS TO SEE IF THIS CONDITION IS OF
  THE FORM 'UN(A)<UN(B)' OR THE FORM 'UN<10'.
                                                    */
/*****************************************************/
```

```
OUT:LAB(8):
        END;
COLUMN:
        POSIT = INDEX(WORK_S,PLACE_C);
        IF POSIT = 0  THEN GO TO OUT1;
        C_WORK_S = '';
        IF POSIT = 1
        THEN COMMA = INDEX(WORK_S,',');
        ELSE DO;
                C_WORK_S = SUBSTR(WORK_S,1,POSIT-1);
                WORK_S = SUBSTR(WORK_S,POSIT);
                COMMA = INDEX(WORK_S,',');
            END;
        IF COMMA = 0
        THEN DO;
                SEMI  = INDEX(WORK_S,';');
                HOLD = SUBSTR(WORK_S,1,SEMI-1);
                WORK_S = '';
                IF C_WORK_S ¬= ''
                THEN C_WORK_S=SUBSTR(C_WORK_S,1,LENGTH(
                                              C_WORK_S)-1)||';' ;
            END;
        ELSE DO;
                HOLD = SUBSTR(WORK_S,1,COMMA-1);
                WORK_S = SUBSTR(WORK_S,COMMA+1);
        END;
        WORK_S = C_WORK_S || WORK_S;
        IF SUBSTR(HOLD,3,1) = '('
        THEN DO;
/*************************************************************/
/*
      HAVING DETERMINED THE CONDITION TO BE OF THE FORM
   UN(A)<UN(B), THE NEXT SEGMENT OF CODING DETERMINES THE
   TWO ARGUMENTS AND THE OPERATOR BETWEEN THEM. CONTROL THEN
   PASSES TO THE SEGMENT OF CODING SET UP TO HANDLE THAT
   OPERATOR.
                                                            */
/*************************************************************/
                ARG = SUBSTR(HOLD,4,1);
                NUM1 = INDEX(ALPHA,ARG);
                P1 = ARG_PTRS(NUM1);
                ARG1 = P1 -> PROB(I);
                HOLD = SUBSTR(HOLD,6);
                OPTR = '';
                DO WHILE (SUBSTR(HOLD,1,1) < 'A');
                    OPTR = OPTR || SUBSTR(HOLD,1,1);
                    HOLD = SUBSTR(HOLD,2);
                END;
                ARG = SUBSTR(HOLD,4,1);
                NUM2 = INDEX(ALPHA,ARG);
                P2 = ARG_PTRS(NUM2);
                ARG2 = P2 -> PROB(I);
/*************************************************************/
/*
      THE OPERATOR IS LESS THAN.
                                                            */
/*************************************************************/
                IF OPTR = '<'
                THEN DO;
                    IF ARG1  ¬< ARG2
                    THEN DO;
                        DIFFER = ARG1 - ARG2 + 1;
                        IF CHOICE(NUM1) = 0 | CHOICE(NUM1)= 2
                        THEN DO;
                            L_L =  MIN(DIFFER,ARG1);
                            U_L =  ARG1;
                            CALL RANDOM(0,L_L,U_L,R_N);
                            ARG1 = ARG1 - R_N;
                            P1 -> PROB(I) = ARG1;
                            IF ARG1 < ARG2
                            THEN GO TO COLUMN;
                            ELSE DIFFER = ARG1 -ARG2 + 1;
```

```
                    END;
                    IF CHOICE(NUM2) = 0 | CHOICE(NUM2)= 1
                    THEN DO;
                        L_L= MIN(DIFFER, 9-ARG2);
                        U_L= 9 - ARG2;
                        CALL RANDOM(0,L_L,U_L,R_N);
                        ARG2 = ARG2 + R_N;
                        P2 -> PROB(I) = ARG2;
                        IF ARG1 < ARG2
                        THEN GO TO COLUMN;
                        ELSE DIFFER = ARG1 - ARG2 + 1;
                    END;
                    DO WHILE(ARG1 ¬< ARG2);
                        FLAG = '0'B;
                        IF ARG1 ¬= 0 & CHOICE(NUM1) < 4
                        THEN DO;
                            FLAG = '1'B;
                            ARG1 = ARG1 -1;
                        END;
                        IF ARG2 ¬= 9 & CHOICE(NUM2) ¬> 3
                        THEN DO;
                            ARG2 = ARG2 + 1;
                            FLAG = '1'B;
                        END;
                        IF ¬FLAG THEN GO TO L_OUT;
                    END;
          L_OUT:P1 ->PROB(I) = ARG1;
                    P2 ->PROB(I) = ARG2;
                END;
                GO TO COLUMN;
            END;
/*****************************************************/
/*
      THE OPERATOR IS GREATER THAN.
                                                     */
/*****************************************************/
            IF OPTR = '>'
            THEN DO;
                IF ARG1 ¬> ARG2
                THEN DO;
                    DIFFER = ARG2 - ARG1 + 1;
                    IF CHOICE(NUM1)= 0 | CHOICE(NUM1) = 1
                    THEN DO;
                        L_L = MIN(DIFFER,9-ARG1);
                        U_L = 9 - ARG1;
                        CALL RANDOM(0,L_L,U_L,R_N);
                        ARG1 = ARG1 + R_N;
                        P1 -> PROB(I) = ARG1;
                        IF ARG1 > ARG2
                        THEN GO TO COLUMN;
                        ELSE DIFFER = ARG2 - ARG1 + 1;
                    END;
                    IF CHOICE(NUM2) = 0 | CHOICE(NUM2)= 2
                    THEN DO;
                        L_L = MIN(DIFFER,ARG2);
                        U_L = ARG2;
                        CALL RANDOM(0,L_L,U_L,R_N);
                        ARG2 = ARG2 - R_N;
                        P2 -> PROB(I) = ARG1;
                        IF ARG1 > ARG2
                        THEN GO TO COLUMN;
                        ELSE DIFFER = ARG1 - ARG2 + 1;
                     END;
                    DO WHILE(ARG1 ¬>  ARG2);
                       FLAG = '0'B;
                       IF ARG1 ¬= 9 & CHOICE(NUM1) ¬> 3
                       THEN DO;
                                FLAG = '1'B;
                                ARG1 = ARG1 + 1;
                            END;
                       IF ARG2 ¬= 0 & CHOICE(NUM2) ¬> 3
                       THEN DO;
```

```
                                    FLAG = '1'B;
                                    ARG2 = ARG2 - 1;
                                END;
                            IF ¬FLAG
                            THEN  GO TO G_OUT;
                        END;
                G_OUT:P1 -> PROB(I) = ARG1;
                        P2 -> PROB(I) = ARG2;
                    END;
                    GO TO COLUMN;
                END;
/*************************************************************/
/*
      THE OPERATOR IS EQUAL.
                                                              */
/*************************************************************/
                IF OPTR = '='
                THEN DO;
                    IF ARG1 ¬= ARG2
                    THEN DO;
                        IF ARG1 > ARG2
                        THEN DO;
                            IF CHOICE(NUM1)=0|CHOICE(NUM1)=2
                            THEN DO;
                                P1 -> PROB(I) = ARG2;
                                GO TO COLUMN;
                            END;
                            IF CHOICE(NUM2)=0|CHOICE(NUM2)=1
                            THEN DO;
                                P2->PROB(I) = ARG1;
                                GO TO COLUMN;
                            END;
                        REDO: FLAG = '0'B;
                            IF CHOICE(NUM1)< 4
                            THEN DO;
                                ARG1 = ARG1 - 1;
                                FLAG = '1'B;
                            END;
                            IF ARG1 ¬= ARG2
                            THEN DO;
                                IF CHOICE(NUM2) < 4
                                THEN DO;
                                    ARG2 = ARG2 + 1;
                                    FLAG = '1'B;
                                END;
                            END;
                            IF ARG1 ¬= ARG2 & FLAG
                            THEN GO TO REDO;
                            P1 ->PROB(I) = ARG1;
                            P2 ->PROB(I) = ARG2;
                            GO TO COLUMN;
                        END;
                        IF ARG1 < ARG2
                        THEN DO;
                            IF CHOICE(NUM1)=0|CHOICE(NUM1)=1
                            THEN DO;
                                P1 -> PROB(I) = ARG2;
                                GO TO COLUMN;
                            END;
                            IF CHOICE(NUM2)=0|CHOICE(NUM2)=2
                            THEN DO;
                                P2 -> PROB(I) = ARG1;
                                GO TO COLUMN;
                            END;
                        REDA:FLAG = '0'B;
                            IF CHOICE(NUM1) < 4
                            THEN DO;
                                ARG1 = ARG1 + 1;
                                FLAG = '1'B;
                            END;
                            IF ARG1 ¬= ARG2
                            THEN DO;
```

```
                                IF CHOICE(NUM2) < 4
                                THEN DO;
                                     ARG2 = ARG2 - 1;
                                     FLAG = '1'B;
                                END;
                           END;
                           IF ARG1 ¬= ARG2 & FLAG
                           THEN GO TO REDA;
                           P1 -> PROB(I) = ARG1;
                           P2 -> PROB(I) = ARG2;
                           GO TO COLUMN;
                      END;
                 END;
                 GO TO COLUMN;
            END;
/*************************************************************/
/*
      THE OPERATOR IS LESS THAN OR EQUAL.
                                                             */
/*************************************************************/
            IF OPTR = '<='
            THEN DO;
                 IF ARG1 > ARG2
                 THEN DO;
                      DIFFER = ARG1 - ARG2;
                      IF CHOICE(NUM1) = 0 | CHOICE(NUM2)= 2
                      THEN DO;
                           L_L = MIN(DIFFER,ARG1);
                           U_L = ARG1;
                           CALL RANDOM(0,L_L,U_L,R_N);
                           P1 -> PROB(I) = ARG1 - R_N;
                           GO TO COLUMN;
                      END;
                      IF CHOICE(NUM2) = 0 | CHOICE(NUM2)= 1
                      THEN DO;
                           L_L = MIN(DIFFER,9-ARG2);
                           U_L = 9 - ARG2;
                           CALL RANDOM(0,L_L,U_L,R_N);
                           P2 -> PROB(I) = ARG2 + R_N;
                           GO TO COLUMN;
                      END;
                      DO WHILE(ARG1 > ARG2);
                           FLAG = '0'B;
                           IF  ARG1 ¬= 0 & CHOICE(NUM1) < 4
                           THEN DO;
                                     FLAG ='1'B;
                                     ARG1 = ARG1 - 1;
                                END;
                           IF ARG2 ¬= 9 & CHOICE(NUM2) < 4
                           THEN DO;
                                     FLAG ='1'B;
                                     ARG2 = ARG2 + 1;
                                END;
                           IF ¬FLAG THEN GO TO LE_OUT;
                      END;
             LE_OUT:  P1 -> PROB(I) = ARG1;
                      P2 -> PROB(I) = ARG2;
                 END;
                 GO TO COLUMN;
            END;
/*************************************************************/
/*
      THE OPERATOR IS GREATER THAN OR EQUAL.
                                                             */
/*************************************************************/
            IF OPTR = '>='
            THEN DO;
                 IF ARG1 < ARG2
                 THEN DO;
                      DIFFER = ARG2 - ARG1;
                      IF CHOICE(NUM1) = 0 | CHOICE(NUM2)= 1
                      THEN DO;
```

```
                        L_L = MIN(DIFFER,9-ARG1);
                        U_L = 9 - ARG1;
                        CALL RANDOM(0,L_L,U_L,R_N);
                        P1 -> PROB(I) = ARG1 + R_N;
                        GO TO COLUMN;
                    END;
                    IF CHOICE(NUM2) = 0 | CHOICE(NUM2)= 2
                    THEN DO;
                        L_L = MIN(DIFFER,ARG2);
                        U_L = ARG2;
                        CALL RANDOM(0,L_L,U_L,R_N);
                        P2 -> PROB(I) = ARG2 - R_N;
                        GO TO COLUMN;
                    END;
                    DO  WHILE(ARG1 < ARG2);
                        FLAG = '0'B;
                        IF ARG1 ¬= 9 & CHOICE(NUM1) < 4
                        THEN DO;
                                FLAG = '1'B;
                                ARG1 = ARG1 + 1;
                            END;
                        IF ARG2 ¬= 0 & CHOICE(NUM2) < 4
                        THEN DO;
                                FLAG = '1'B;
                                ARG2 = ARG2 - 1;
                            END;
                        IF ¬FLAG THEN GO TO GE_OUT;
                    END;
          GE_OUT:   P1 -> PROB(I) = ARG1;
                    P2 -> PROB(I) = ARG2;
                END;
                GO TO COLUMN;
            END;
/*************************************************************/
/*
      THE OPERATOR IS DIVISIBLE BY OR MULTIPLE OF.
                                                             */
/*************************************************************/
            IF OPTR = '//' | OPTR = '**'
            THEN DO;
                REMAIN = MOD(ARG1,ARG2);
                IF REMAIN ¬= 0
                THEN DO;
                    FLAG = '0'B;
                    IF ARG1 > ARG2
                    THEN DO;
                        IF CHOICE(NUM1)=0|CHOICE(NUM1)=2
                        THEN DO;
                            P1 -> PROB(I) = ARG1-REMAIN;
                            CHOICE(NUM1) = 4;
                            FLAG = '1'B;
                        END;
                    IF CHOICE(NUM1) = 1 & (ARG1 + ARG2
                                           -REMAIN) < 10
                        THEN DO;
                            P1 -> PROB(I) = ARG1 +
                                            (ARG2-REMAIN)
                            CHOICE(NUM1) = 4;
                            FLAG = '1'B;
                        END;
                    END;
                    IF ARG2 > ARG1
                    THEN DO;
                        IF CHOICE(NUM1)=0|CHOICE(NUM1)=1
                        THEN DO;
                            P1 ->PROB(I) = ARG1 +
                                           (ARG2-REMAIN);
                            CHOICE(NUM1) = 4;
                            FLAG = '1'B;
                        END;
                        IF CHOICE(NUM1) = 2
                        THEN DO;
```

```
                                P1 -> PROB(I) = ARG1 - REMAIN
                                CHOICE(NUM1) = 4;
                                FLAG = '1'B;
                            END;
                        END;
                        IF ¬FLAG
                        THEN DO;
                            IF CHOICE(NUM1) < 4
                            THEN DO;
                                P1 -> PROB(I) = ARG1 - REMAIN
                                CHOICE(NUM1) = 4;
                                FLAG = '1'B;
                            END;
                        END;
                        IF ¬FLAG
                        THEN DO;
                            IF CHOICE(NUM2) < 4
                            THEN DO;
                                P2 -> PROB(I) = ARG1;
                                CHOICE(NUM2) = 4;
                            END;
                        END;
                    END;
                     GO TO COLUMN;
                END;
            END;
        ELSE DO;
/*****************************************************/
/*
      IF THE CONDITION IS DETERMINED TO BE OF THE FORM
   'UN<10', THEN CONTROL IS PASSED TO THIS POINT. THE SUM OF
   THE PLACE VALUE UNDER CONSIDERATION IS DETERMINED AS IS
   ANY CARRY FROM A PREVIOUS PLACE VALUE. THEN THE OPERATOR
   IS DETERMINED AND CONTROL PASSED TO THE SEGMENT WHICH
   HANDLES THAT OPERATOR.
                                                      */
/*****************************************************/
                OPTR = '';
                HOLD = SUBSTR(HOLD,3);
                DO WHILE(SUBSTR(HOLD,1,1) < 'A');
                  OPTR = OPTR || SUBSTR(HOLD,1,1);
                  HOLD = SUBSTR(HOLD,2);
                END;
                COMP = HOLD;
                CARRY = 0;
                IF  I > 1
                THEN DO  K = 1 TO ARG_PTR->NR;
                    P_PTR = ARG_PTRS(INDEX(ALPHA,ARG_PTR->ARG
                    CARRY = CARRY + P_PTR->PROB(I);
                END;
                CARRY = CARRY/10;
                TOTAL = 0;
                DO K= 1 TO ARG_PTR -> NR;
                    P_PTR = ARG_PTRS(INDEX(ALPHA,ARG_PTR ->
                                                ARGUE(K)));
                    TOTAL = TOTAL + P_PTR -> PROB(I);
                END;
                TOTAL = TOTAL + CARRY;
/*****************************************************/
/*
      THE OPERATOR IS LESS THAN.
                                                      */
/*****************************************************/
                IF OPTR = '<'
                THEN DO;
          L_REDO1:FLAG = '0'B;
                    DO K=1 TO ARG_PTR->NR WHILE(TOTAL¬<COMP);
                        NUM=INDEX(ALPHA,ARG_PTR->ARGUE(K));
                        P_PTR = ARG_PTRS(NUM);
                        DIFFER = TOTAL - COMP + 1;
                        IF DIFFER > P_PTR-> PROB(I)
                        THEN L_L = 1;
```

```
                    ELSE L_L = DIFFER;
                    U_L = P_PTR -> PROB(I);
                    IF CHOICE(NUM)=0 | CHOICE(NUM)=2
                    THEN IF P_PTR->PROB(I) > 0
                    THEN DO;
                         CALL RANDOM(0,L_L,U_L,R_N);
                         P_PTR->PROB(I)=P_PTR->PROB(I)-R_N
                         TOTAL = TOTAL - R_N;
                         IF P_PTR -> PROB(I) = 0
                         THEN CHOICE(NUM) = 3;
                         FLAG ='1'B;
                    END;
               END;
               IF(TOTAL¬<COMP)&FLAG THEN GO TO L_REDO1;
     L_REDO2:IF(TOTAL¬<COMP)&¬FLAG
               THEN DO;
                    DO K=1 TO ARG_PTR->NR WHILE
                                        (TOTAL ¬<COMP);
                         NUM=INDEX(ALPHA,ARG_PTR->ARGUE(K)
                         P_PTR = ARG_PTRS(NUM);
                         IF CHOICE(NUM) = 3
                         THEN IF P_PTR->PROB(I) > 0
                              THEN DO;
                                   P_PTR->PROB(I)=P_PTR ->
                                                  PROB(I) - 1;
                                   TOTAL = TOTAL-1;
                                   FLAG = '1'B;
                              END;
                     END;
                     IF FLAG
                     THEN DO;
                              FLAG = '0'B;
                              GO TO L_REDO2;
                          END;
               END;
     L_REDO3:IF (TOTAL¬<COMP) & ¬FLAG
               THEN DO;
                    DO K = 1 TO ARG_PTR->NR WHILE(TOTAL¬<
                         NUM = INDEX(ALPHA,ARG_PTR->
                                                ARGUE(K));
                         P_PTR = ARG_PTRS(NUM);
                         IF CHOICE(NUM) < 4
                         THEN IF P_PTR ->PROB(I) >0
                              THEN DO;
                                   P_PTR->PROB(I) = P_PTR->
                                                  PROB(I)-1;
                                   TOTAL = TOTAL-1;
                                   FLAG = '1'B;
                              END;
                    END;
                    IF TOTAL ¬< COMP & FLAG
                    THEN DO;
                         FLAG = '0'B;
                         GO TO L_REDO3;
                    END;
               END;
               GO TO COLUMN;
          END;
/*****************************************************/
/*
     THE OPERATOR IS GREATER THAN.
                                                      */
/*****************************************************/
          IF OPTR = '>'
          THEN DO;
     G_REDO1:FLAG = '0'B;
               DO K=1 TO ARG_PTR->NR WHILE(TOTAL¬>COMP);
                    NUM=INDEX(ALPHA,ARG_PTR->ARGUE(K));
                    P_PTR = ARG_PTRS(NUM);
                    DIFFER = COMP - TOTAL + 1;
                    IF DIFFER < 9-P_PTR -> PROB(I)
                    THEN U_L = 9- P_PTR -> PROB(I);
```

```
                    ELSE U_L = DIFFER;
                    L_L = DIFFER;
                    IF CHOICE(NUM)=0 | CHOICE(NUM)=1
                    THEN IF P_PTR->PROB(I) < 9
                    THEN DO;
                         CALL RANDOM(0,L_L,U_L,R_N);
                         P_PTR->PROB(I)=P_PTR->PROB(I)+R_N
                         TOTAL = TOTAL + R_N;
                         IF P_PTR -> PROB(I) = 9
                         THEN CHOICE(NUM) = 3;
                         FLAG = '1'B;
                    END;
               END;
               IF TOTAL ¬> COMP & FLAG
               THEN GO TO G_REDO1;
     G_REDO2:IF TOTAL ¬> COMP & ¬FLAG
               THEN DO;
                    DO K=1 TO ARG_PTR->NR WHILE
                                             (TOTAL¬>COMP);
                         NUM=INDEX(ALPHA,ARG_PTR->
                                             ARGUE(K));
                         P_PTR = ARG_PTRS(NUM);
                         IF CHOICE(NUM) = 3
                         THEN IF P_PTR->PROB(I) < 9
                              THEN DO;
                                   P_PTR->PROB(I)=P_PTR ->
                                             PROB(I)+1;
                                   TOTAL = TOTAL + 1;
                                   FLAG = '1'B;
                              END;
                    END;
                    IF FLAG
                    THEN DO;
                              FLAG = '0'B;
                              GO TO G_REDO2;
                         END;
               END;
     G_REDO3:IF (TOTAL¬>COMP) & ¬FLAG
               THEN DO;
                    DO K=1 TO ARG_PTR->NR WHILE(TOTAL¬> C
                         NUM = INDEX(ALPHA,ARG_PTR->
                                             ARGUE(K));
                         P_PTR = ARG_PTRS(NUM);
                         IF CHOICE(NUM) < 4
                         THEN IF P_PTR-> PROB(I) < 9
                              THEN DO;
                                   P_PTR->PROB(I) = P_PTR->
                                             PROB(I)+1;
                                   TOTAL = TOTAL+1;
                                   FLAG = '1'B;
                              END;
                    END;
                    IF (TOTAL¬>COMP) & FLAG
                    THEN DO;
                         FLAG = '0'B;
                         GO TO G_REDO3;
                    END;
               END;
               GO TO COLUMN;
          END;
/*************************************************************/
/*
      THE OPERATOR IS LESS THAN OR EQUAL.
                                                             */
/*************************************************************/
          IF OPTR = '<='
          THEN DO;
     LE_REDO1:FLAG = '0'B;
               DO K = 1 TO ARG_PTR->NR WHILE(TOTAL>COMP)
                    NUM = INDEX(ALPHA,ARG_PTR->ARGUE(K));
                    P_PTR = ARG_PTRS(NUM);
                    DIFFER = TOTAL - COMP;
```

```
                    IF DIFFER > P_PTR->PROB(I)
                    THEN L_L = 1;
                    ELSE L_L = DIFFER;
                    U_L = P_PTR -> PROB(I);
                    IF CHOICE(NUM)=0 | CHOICE(NUM)=2
                    THEN IF P_PTR->PROB(I)> 0
                    THEN DO;
                         CALL RANDOM(0,L_L,U_L,R_N);
                         P_PTR->PROB(I)=P_PTR->PROB(I)-R_N
                         TOTAL = TOTAL - R_N;
                         IF P_PTR-> PROB(I) = 0
                         THEN CHOICE(NUM) = 3;
                         FLAG = '1'B;
                    END;
               END;
               IF (TOTAL>COMP)&FLAG THEN GO TO LE_REDO1;
LE_REDO2:IF (TOTAL>COMP)&¬FLAG
               THEN DO;
                    DO K=1 TO ARG_PTR->NR WHILE
                                        (TOTAL > COMP);
                         NUM=INDEX(ALPHA,ARG_PTR->ARGUE(K)
                         P_PTR = ARG_PTRS(NUM);
                         IF CHOICE(NUM) = 3
                         THEN IF P_PTR->PROB(I) > 0
                              THEN DO;
                                   P_PTR->PROB(I)=P_PTR ->
                                                PROB(I) - 1;
                                   FLAG = '1'B;
                                   TOTAL = TOTAL -1;
                              END;
                    END;
                    IF FLAG
                    THEN DO;
                              FLAG = '0'B;
                              GO TO L_REDO2;
                         END;
               END;
LE_REDO3:IF(TOTAL>COMP) & ¬FLAG
               THEN DO;
                    DO K = 1 TO ARG_PTR->NR WHILE(TOTAL>C
                         NUM = INDEX(ALPHA,ARG_PTR->
                                                ARGUE(K));
                         P_PTR = ARG_PTRS(NUM);
                         IF CHOICE(NUM) < 4
                         THEN IF P_PTR -> PROB(I) >0
                              THEN DO;
                                   P_PTR->PROB(I) = P_PTR->
                                                PROB(I)-1;
                                   TOTAL = TOTAL-1;
                                   FLAG = '1'B;
                              END;
                    END;
                    IF(TOTAL>COMP)& FLAG
                    THEN DO;
                         FLAG = '0'B;
                         GO TO LE_REDO3;
                    END;
               END;
               GO TO COLUMN;
          END;
/*****************************************************************/
/*
     THE OPERATOR IS GREATER THAN OR EQUAL.                      */
/*****************************************************************/
          IF OPTR = '>='
          THEN DO;
GE_REDO1:FLAG = '0'B;
               DO K=1 TO ARG_PTR->NR WHILE(TOTAL<COMP);
                    NUM = INDEX(ALPHA,ARG_PTR->ARGUE(K));
                    P_PTR = ARG_PTRS(NUM);
                    DIFFER = COMP - TOTAL;
```

```
                    IF DIFFER < 9-P_PTR-> PROB(I)
                    THEN U_L = 9 - P_PTR -> PROB(I);
                    ELSE U_L = DIFFER;
                    L_L = DIFFER;
                    IF CHOICE(NUM)=0 | CHOICE(NUM)=1
                    THEN IF P_PTR -> PROB(I) < 9
                         THEN DO;
                              CALL RANDOM(0,L_L,U_L,R_N);
                              P_PTR->PROB(I)=P_PTR->PROB(I)
                                             +R_N;
                              IF P_PTR -> PROB(I) = 9
                              THEN CHOICE(NUM)=3;
                              FLAG = '1'B;
                              TOTAL = TOTAL + R_N;
                         END;
               END;
               IF TOTAL < COMP & FLAG
               THEN GO TO GE_REDO1;
GE_REDO2:IF TOTAL < COMP & ¬FLAG
               THEN DO;
                    DO K=1 TO ARG_PTR->NR
                       WHILE(TOTAL<COMP);
                         NUM = INDEX(ALPHA,ARG_PTR->
                                          ARGUE(K));
                         P_PTR = ARG_PTRS(NUM);
                         IF CHOICE(NUM) = 3
                         THEN IF P_PTR-> PROB(I) < 9
                         THEN DO;
                              P_PTR->PROB(I)=P_PTR ->
                                          PROB(I)+1;
                                TOTAL = TOTAL + 1;
                                FLAG = '1'B;
                              END;
                    END;
                    IF FLAG
                    THEN DO;
                              FLAG ='0'B;
                              GO TO G_REDO2;
                    END;
               END;
GE_REDO3:IF (TOTAL<COMP) & ¬FLAG
               THEN DO;
                    DO K = 1 TO ARG_PTR->NR WHILE
                                        (TOTAL<COMP);
                         NUM = INDEX(ALPHA,ARG_PTRS ->
                                          ARGUE(K));
                         P_PTR = ARG_PTRS(NUM);
                         IF CHOICE(NUM) < 4
                         THEN IF P_PTR -> PROB(I) < 0
                              THEN DO;
                                   P_PTR -> PROB(I) = P_
                                        ->PROB(I)+1;
                                TOTAL = TOTAL + 1;
                                FLAG = '1'B;
                              END;
                    END;
                    IF TOTAL < COMP & FLAG
                    THEN DO;
                         FLAG = '0'B;
                         GO TO GE_REDO3;
                    END;
               END;
               GO TO COLUMN;
          END;
/*****************************************************/
/*
     THE OPERATOR IS EQUAL.                           */
/*****************************************************/
          IF OPTR = '='
          THEN DO;
               IF TOTAL > COMP
```

```
    THEN DO;
E_REDO1:FLAG ='0'B;
        DO K=1 TO ARG_PTR->NR WHILE(TOTAL¬=CO
            NUM=INDEX(ALPHA,ARG_PTR->ARGUE(K)
            P_PTR = ARG_PTRS(NUM);
            DIFFER = TOTAL - COMP;
            IF DIFFER>P_PTR -> PROB(I)
            THEN L_L = I;
            ELSE L_L = DIFFER;
            U_L = DIFFER;
            IF CHOICE(NUM)=0 | CHOICE(NUM)=2
            THEN IF P_PTR->PROB(I) > 0
            THEN DO;
                CALL RANDOM(0,L_L,U_L,R_N);
                P_PTR -> PROB(I) = P_PTR->
                                   PROB(I)-R_N;
                TOTAL = TOTAL - R_N;
                IF P_PTR -> PROB(I) = 0
                THEN CHOICE(NUM) = 4;
                FLAG = '1'B;
            END;
        END;
        IF TOTAL ¬= COMP & FLAG
        THEN GO TO E_REDO1;
E_REDO2:IF TOTAL ¬= COMP & ¬FLAG
        THEN DO;
                DO K=1 TO ARG_PTR -> NR
                    WHILE(TOTAL ¬= COMP);
                    NUM = INDEX(ALPHA,ARG_PTR
                                -> ARGUE(K));
                    P_PTR = ARG_PTRS(NUM);
                    IF CHOICE(NUM) = 3
                    THEN IF P_PTR->PROB(I)> 0
                        THEN DO;
                            P_PTR->PROB(I) =
                            P_PTR->PROB(I)-1
                            TOTAL = TOTAL -1
                            FLAG = '1'B;
                        END;
                  END;
                IF FLAG
                THEN DO;
                    FLAG = '0'B;
                    GO TO E_REDO2;
                END;
            END;
E_REDO3:IF (TOTAL¬=COMP) & ¬FLAG
        THEN DO;
            DO K = 1 TO ARG_PTR->NR
                WHILE(TOTAL¬=COMP);
            NUM = INDEX(ALPHA,ARG_PTR->
                                ARGUE(K));
            P_PTR = ARG_PTRS(NUM);
            IF CHOICE(NUM) < 4
            THEN IF P_PTR -> PROB(I) >0
                THEN DO;
                    P_PTR->PROB(I) =
                      P_PTR->PROB(I)-1;
                    TOTAL = TOTAL-1;
                    FLAG = '1'B;
                END;
            END;
            IF(TOTAL ¬= COMP)& FLAG
            THEN DO;
                FLAG = '0'B;
                GO TO E_REDO3;
            END;
        END;
    END;
    IF TOTAL < COMP
    THEN DO;
E_REDO4:FLAG = '0'B;
```

```
                DO K=1 TO ARG_PTR->NR
                     WHILE(TOTAL < COMP);
                   NUM=INDEX(ALPHA,ARG_PTR->ARGUE(K)
                   P_PTR = ARG_PTRS(NUM);
                   DIFFER = COMP - TOTAL;
                   IF 9-P_PTR->PROB(I) > DIFFER
                   THEN U_L = DIFFER;
                   ELSE U_L = 9 - P_PTR-> PROB(I);
                   L_L = 1;
                   IF CHOICE(NUM)=0 | CHOICE(NUM)=1
                   THEN IF P_PTR->PROB(I) < 9
                   THEN DO;
                        CALL RANDOM(0,L_L,U_L,R_N);
                        P_PTR -> PROB(I)= P_PTR ->
                                          PROB(I)+R_N;
                        TOTAL = TOTAL + R_N;
                        IF P_PTR -> PROB(I) = 9
                        THEN CHOICE(NUM) = 3;
                        FLAG = '1'B;
                   END;
              END;
              IF TOTAL<COMP & FLAG
              THEN GO TO E_REDO4;
E_REDO5:IF TOTAL < COMP & ¬FLAG
              THEN DO;
                   DO K = 1 TO ARG_PTR -> NR
                        WHILE(TOTAL < COMP);
                        NUM = INDEX(ALPHA,ARG_PTR
                                    -> ARGUE(K));
                        P_PTR = ARG_PTRS(NUM);
                        IF CHOICE(NUM) = 3
                        THEN IF P_PTR->PROB(I)< 9
                             THEN DO;
                                  P_PTR->PROB(I) =
                                  P_PTR->PROB(I)+1
                                  TOTAL = TOTAL+1;
                                  FLAG = '1'B;
                             END;
                   END;
                   IF FLAG
                   THEN DO;
                        FLAG = '0'B;
                        GO TO E_REDO2;
                   END;
              END;
E_REDO6:IF (TOTAL¬=COMP) & ¬FLAG
              THEN DO;
                   DO K = 1 TO ARG_PTR-> NR
                        WHILE(TOTAL ¬= COMP);
                   NUM = INDEX(ALPHA,ARG_PTR->
                                     ARGUE(K));
                   P_PTR = ARG_PTRS(NUM);
                   IF CHOICE(NUM) < 4
                   THEN IF P_PTR -> PROB(I) <9
                        THEN DO;
                             P_PTR->PROB(I) =
                               P_PTR->PROB(I)+1;
                             TOTAL = TOTAL+1;
                             FLAG = '1'B;
                        END;
                   END;
                   IF TOTAL ¬= COMP & FLAG
                   THEN DO;
                        FLAG = '0'B;
                        GO TO E_REDO6;
                   END;
              END;
         END;
         GO TO COLUMN;
    END;
END;
```

```
 /*****************************************************************/
 /*
       NO CODING HAS YET BEEN WRITTEN TO HANDLE CONDITIONS
   OF THE FORM 'UN//8' OR 'UN**8' SINCE IT COULD NOT BE SEEN
   WHERE THESE WOULD HAVE ANY USE IN DESCRIBING A PROBLEM.
   THE CODE SHOULD BE INCLUDED FOR COMPLETENESS PRIOR TO THE
   FINAL TESTING OT THE SYSTEM.
                                                                 */
 /*****************************************************************/
OUT1:END;
 /*****************************************************************/
 /*
       WHEN ALL ARGUMENTS HAVE BEEN COMPLETELY GENERATED AND
   ALL CONDITIONS OF GROUP 1 AND GROUP 2 HANDLED, CONTROL
   PASSES OUT OF THE DO LOOP BEGINNING AT N1 AND A CHECK IS
   MADE FOR ANY CONDITIONS BELONGING TO GROUP 3. IF ANY ARE
   ENCOUNTERED, THE TWO ARGUMENTS IN QUESTION ARE DETERMINED
   AND THEN CONTROL IS PASSED TO THE SEGMENT OF CODING WHICH
   HANDLES THE OPERATOR RELATING THEN.
                                                                 */
 /*****************************************************************/
ANYMORE:
   IF WORK_S = ''
   THEN GO TO OUT3;
   COMMA = INDEX(WORK_S,',');
   IF COMMA = 0
   THEN DO;
        SEMI = INDEX(WORK_S,';');
        HOLD = SUBSTR(WORK_S,1,SEMI-1);
        WORK_S = '';
   END;
   ELSE DO;
        HOLD = SUBSTR(WORK_S,1,COMMA-1);
        WORK_S = SUBSTR(WORK_S,COMMA+1);
   END;
   CARG1 = SUBSTR(HOLD,1,1);
   HOLD = SUBSTR(HOLD,2);
   OPTR = '';
   DO WHILE(SUBSTR(HOLD,1,1) <'A');
        OPTR = OPTR || SUBSTR(HOLD,1,1);
        HOLD = SUBSTR(HOLD,2);
   END;
   CARG2 = SUBSTR(HOLD,1,1);
   NUM1 = INDEX(ALPHA,CARG1);
   NUM2 = INDEX(ALPHA,CARG2);
   P_PTR = ARG_PTRS(NUM2);
   ARG2 = 0;
   DO MM = 1 TO P_PTR -> DIGETS;
        ARG2 = ARG2 + P_PTR -> PROB(MM)*(10**(MM-1));
   END;
   P_PTR = ARG_PTRS(NUM1);
   ARG1 = 0;
   DO MM = 1 TO P_PTR -> DIGETS;
        ARG1 = ARG1 + P_PTR -> PROB(MM)* (10**(MM-1));
   END;
 /*****************************************************************/
 /*
       THE OPERATOR IS LESS THAN.
                                                                 */
 /*****************************************************************/
   IF  OPTR = '<'
   THEN DO;
        IF ARG1 ¬< ARG2
        THEN DO;
             DIFFER = ARG1 - ARG2 + 1;
             IF ARG1 > DIFFER
             THEN DO;
                  U_L = ARG1;
                  L_L = DIFFER;
                  CALL RANDOM(0,L_L,U_L,R_N);
                  ARG1 = ARG1 - R_N;  FLAG = '1'B ;
                  GO TO REPLACE;
```

```
            END;
            ELSE DO;
                 P_PTR = ARG_PTRS(NUM2);
                 U_L = 10 ** P_PTR->DIGETS -(ARG2 + 1);
                 L_L = DIFFER;
                 CALL RANDOM(0,L_L,U_L,R_N);
                 ARG2= ARG2 + R_N;  FLAG = '0'B;
                 GO TO REPLACE;
            END;
       GO TO ANYMORE;
       END;
  END;
/*****************************************************************/
/*
       THE OPERATOR IS GREATER THAN.
                                                                 */
/*****************************************************************/
  IF  OPTR = '>'
  THEN DO;
       IF ARG1 ¬> ARG2
       THEN DO;
            DIFFER = ARG2 - ARG1 + 1;
            IF 10 ** P_PTR->DIGETS - (ARG1 + 1) > DIFFER
            THEN DO;
                 U_L = 10 ** P_PTR->DIGETS -(ARG1 + 1);
                 L_L = DIFFER;
                 CALL RANDOM(0,L_L,U_L,R_N);
                 ARG1 = ARG1 + R_N; FLAG = '1'B;
                 GO TO REPLACE;
            END;
            ELSE DO;
                 P_PTR = ARG_PTRS(NUM2);
                 U_L = ARG2;
                 L_L = DIFFER;
                 CALL RANDOM(0,L_L,U_L,R_N);
                 ARG2 = ARG2 - R_N; FLAG = '0'B;
                 GO TO REPLACE;
            END;
       END;
       GO TO ANYMORE;
  END;
/*****************************************************************/
/*
       THE OPERATOR IS LESS THAN OR EQUAL.
                                                                 */
/*****************************************************************/
  IF OPTR = '<='
  THEN DO;
       IF ARG1 > ARG2
       THEN DO;
            DIFFER = ARG1 - ARG2;
            IF ARG1 > DIFFER
            THEN DO;
                 U_L = ARG1;
                 L_L = DIFFER;
                 CALL RANDOM(0,L_L,U_L,R_N);
                 ARG1 = ARG1 - R_N; FLAG = '1'B;
                 GO TO REPLACE;
            END;
            ELSE DO;
                 P_PTR = ARG_PTRS(NUM2);
                 U_L = 10** P_PTR->DIGETS -(ARG1+1);
                 L_L = DIFFER;
                 CALL RANDOM(0,L_L,U_L,R_N);
                 ARG2 = ARG2 + R_N; FLAG = '0'B;
                 GO TO REPLACE;
            END;
       END;
       GO TO ANYMORE;
  END;
```

```
/*************************************************************/
/*
      THE OPERATOR IS GREATER THAN OR EQUAL.
                                                             */
/*************************************************************/
  IF  OPTR = '>='
  THEN DO;
       IF ARG1 < ARG2
       THEN DO;
            DIFFER = ARG2 - ARG1;
            IF 10 ** P_PTR->DIGETS - (ARG1 +1) > DIFFER
            THEN DO;
                 U_L = 10 ** P_PTR->DIGETS -(ARG1 + 1);
                 L_L = DIFFER;
                 CALL RANDOM(0,L_L,U_L,R_N);
                 ARG1 = ARG1 + R_N; FLAG = '1'B;
                 GO TO REPLACE;
            END;
            ELSE DO;
                 P_PTR = ARG_PTRS(NUM2);
                 U_L = ARG2;
                 L_L = DIFFER;
                 CALL RANDOM(0,L_L,U_L,R_N);
                 ARG2 = ARG2 - R_N; FLAG = '0'B;
                 GO TO REPLACE;
            END;
       END;
       GO TO ANYMORE;
  END;
/*************************************************************/
/*
      THE OPERATOR IS DIVISIBLE BY OR MULTIPLE OF.
                                                             */
/*************************************************************/
  IF OPTR = '//' | OPTR = '**'
  THEN DO;
       REMAIN = MOD(ARG1,ARG2);
       IF REMAIN ¬= 0
       THEN DO;
            IF REMAIN < ARG2-REMAIN
            THEN IF ARG1 - REMAIN > 0
                 THEN ARG1 = ARG1 - REMAIN;
                 ELSE ARG1 = ARG1 + (ARG2 - REMAIN);
            ELSE IF ARG1 + (ARG2 - REMAIN)< 10 ** P_PTR->DIGE
                 THEN ARG1 = ARG1 + (ARG2 - REMAIN);
                 ELSE ARG1 = ARG1 - REMAIN;
            FLAG ='1'B;
                 GO TO REPLACE;
       END;
       GO TO ANYMORE;
  END;
/*************************************************************/
/*
      THE OPERATOR IS EQUAL.
                                                             */
/*************************************************************/
  IF OPTR = '='
  THEN DO;
       IF ARG1 ¬= ARG2
       THEN DO;
            ARG1 = ARG2;
            FLAG = '1'B; GO TO REPLACE;
       END;
       GO TO ANYMORE;
  END;
/*************************************************************/
/*
      AFTER CHANGE HAS BEEN MADE IN NUMBER TO SATISFY THE
  CONDITION, THE CHANGES ARE MADE IN THE STORAGE LOCATIONS
  OF THE NUMBER, STARTING AT  REPLACE.
                                                             */
/*************************************************************/
```

```
REPLACE:MN = 10;
        DO MM= 1 TO P_PTR -> DIGETS;
             IF FLAG
             THEN DO;
                  P_PTR -> PROB(MM) =  MOD(ARG1,MN);
                  ARG1 = ARG1/MN;
             END;
             ELSE DO;
                  P_PTR -> PROB(MM) =  MOD(ARG2,MN);
                  ARG2 = ARG2/MN ;
             END;
        END;
        GO TO ANYMORE;
 /*************************************************************/
 /*
        WHENEVER WORK_S BECOMES NULL, CONTRCL IS PASSED TO
    'FILL' WHICH GENERATES NUMBERS TO FILL UP THE REMAINING
    DIGITS NOT YET GENERATED.
                                                             */
 /*************************************************************/
FILL:U_L = 9;
    L_L = 0;
    DO K = LAST TO G_DIG;
        DO L = 1 TO ARG_PTR -> NR;
             ARG = ARG_PTR -> ARGUE(L);
             NUM = INDEX(ALPHA,ARG);
             P_PTR = ARG_PTRS(NUM);
             IF P_PTR -> DIGETS < K
             THEN GO TO OUT2;
             ELSE DO;
                  P_PTR -> PROB(K) = R_N;
             END;
   OUT2:END;
   END;
END PRO_GEN;
```

```
PROBSOL: PROC(PROB_S);

/*****************************************************/
/*
     THE ROUTINE PROBSOL IS USED TO CALCULATE AN ANSWER TO
  TO THE PROBLEM PRODUCED BY PRO_GEN. IT OPERATES ON THE
  POLISH EQUIVALENT OF THE PROBLEM FORMAT,PASSED AS A
  PARAMETER. TWO OTHER ROUTINES F_SCAN AND R_SCAN ARE CALL-
  ED TO ASSIST PROBSOL. IT ALLOCATES STORAGE FOR TEMPORARY
  ANSWERS GENERATED IN SOLVING THE PROBLEM AND PLACES THESE
  TEMPORARY ANSWERS IN THE STORAGE LOCATIONS ALLOCATED. THE
  MAIN VARIABLES USED IN PROBSOL ARE:
       PROB_S - POLISH EXPRESSION OF THE PROBLEM.
       POSIT & PLACE - KEEP TRACK OF LOCATION IN PROB_S.
       ARG1,ARG2 - THE TWO ARGUEMENTS TO BE OPERATED ON.
       DIG1,DIG2 - THE SIZE OF ARG1 AND ARG2 RESPECTIVELY.
       PTR1,PTR2 - POINT TO STORAGE LOCATION OF ARG1 AND
                   ARG2 RESPECTIVELY.
       ANS_PTR - POINTER TO STORAGE LOCATION OF ANSWER.
       CAR_PTR - POINTER TO STORAGE LOCATION OF ANY CARRIES
                 GENERATED IN OBTAINING THE ANSWER.
       OPER - THE OPERATION TO BE PERFORMED ON ARG1 & ARG2.
                                                        */
/*****************************************************/

  DCL (ANS_PTR,CAR_PTR) PTR ;
  DCL(T_PTR,TARG_PTR) PTR;
  DCL PROB_P PTR;
  DCL(POSIT,PLACE) FIXED BIN(15);
  DCL PROB_S CHAR(240) VARYING;
  DCL F_SCAN ENTRY(CHAR(240)VARYING) RETURNS(CHAR(1));
  DCL OPER CHAR(1);
  DCL TEMP_C CHAR(9) VARYING;
  DCL (TYPE,ARG_C) CHAR(1);
  DCL NR_ARG FIXED BIN(15);
  DCL(DIG_1,DIG_2) FIXED BIN(15);
  DCL(ARG1,ARG2) CHAR(3) VARYING;
  DCL(PTR_1,PTR_2) PTR;
  DCL(COUNT) FIXED BIN(15);
  DCL FLAG BIT(1);
  ALLOCATE TEMP_NODE;
  TEMP_PTR = TNP;
  T_PTR = TEMP_PTR;
  T_PTR->T_A_P = NULL;
  POSIT = 0; T_PTR  = TEMP_PTR; FLAG = '0'B;
  TEMP  = 1; OPER = '';
/*****************************************************/
/*
  F_SCAN IS CALLED TO LOCATE OPERATOR.
                                                        */
/*****************************************************/

DU:OPER = F_SCAN(PROB_S);

/*****************************************************/
/*
  WHEN THE ';' IS DETECTED IT INDICATES THAT THE END OF
  PROB_S HAS BEEN REACHED
                                                        */
/*****************************************************/

  IF OPER = ';' THEN GO TO FINIS;

/*****************************************************/
/*
     WHEN THE '=' SIGN IS ENCOUNTERED IT MEANS,WITH THE
  PRESENT LIMITATION ON FORMAT,THAT THE SOLUTION HAS BEEN
  FOUND. THE COMPUTER'S ANSWER IS THEN STORED. SINCE OTHER
  FORMATS ARE INTENDED THE SCAN OF PROB_S IS NOT TERMINATED
                                                        */
/*****************************************************/
```

```
   IF OPER = '='
   THEN DO;
        ANSWER = '';
        DO I = 1 TO ANS_PTR-> DIGETS;
             TEMP_C= ANS_PTR->PROB(I);
             ARG_C = SUBSTR(TEMP_C,9,1);
             ANSWER = ARG_C || ANSWER;
             M_ANS_C(I) = ARG_C;
        END;
        GO TO DU;
   END;
 /*************************************************************/
 /*
                TEMPORARY STORAGE IS ALLOCATED TO HOLD THE
   ANSWER JUST CALCULATED FOR THE PERVIOUS OPERATOR
                                                              */
 /*************************************************************/
   IF FLAG
   THEN DO;
             ALLOCATE TEMP_NODE;
             T_PTR->T_P=TNP;
             SIZE = ANS_PTR->DIGETS;
             ALLOCATE PROBLEM;
             T_PTR  = T_PTR->T_P;
             T_PTR -> T_A_P = PP;
             TEMP = TEMP +1;
             PROB_P = T_PTR -> T_A_P;
             PROB_P->PROB = ANS_PTR->PROB;
        END;
   TEMP_C = TEMP;
   DO WHILE(SUBSTR(TEMP_C,1,1)= ' ');
        TEMP_C = SUBSTR(TEMP_C,2);
   END;
 /*************************************************************/
 /*
        R_SCAN IS CALLED TO DETERMINE THE ARGUMENTS TO BE
   USED BY THE OPERATOR. IF THE FIRST CHARACTER IS A '#'
   THEN THE ARGUMENT IS A NUMBER CREATED BY PRO_GEN AND THE
   REST OF THE ARGUMENT INDICATES WHERE THE POINTER TO ITS
   LOCATION IS LOCATED IN THE ARRAY ARG_PTRS. IF THE FIRST
   CHARACTER IS AN '@' THEN THE ARGUMENT IS A PREVIOUSLY
   DETERMINED PARTIAL ANSWER AND THE REST OF THE ARGUMENT
   TELLS WHICH PARTIAL ANSWER.
                                                              */
 /*************************************************************/
   CALL R_SCAN(ARG1,ARG2);
   TYPE =  SUBSTR(ARG1,1,1);
   ARG_C = SUBSTR(ARG1,2);
   COUNT = 0;
   GO TO TYPEQ;
AGAIN:PTR_1 = TARG_PTR;
   TYPE = SUBSTR(ARG2,1,1);
   ARG_C =SUBSTR(ARG2,2);
TYPEQ:COUNT = COUNT + 1;
   IF TYPE = '@'
   THEN DO;
             TARG_PTR = TEMP_PTR;
             NR_ARG = ARG_C;
             DO I = 1 TO NR_ARG;
                TARG_PTR = TARG_PTR->T_P;
             END;
             TARG_PTR = TARG_PTR->T_A_P;
      END;
   IF  TYPE = '#'
   THEN DO;
             NR_ARG = ARG_C;
             TARG_PTR = ARG_PTRS(NR_ARG);
        END;
   IF COUNT = 1 THEN GO TO AGAIN;
   PTR_2 = TARG_PTR;
```

```
/*************************************************************/
/*
      ONCE THE TWO ARGUMENTS HAVE BEEN OBTAINED AND PTR1
   AND PTR2 SET TO POINT TO THEIR STORAGE LOCATIONS THEN THE
   ROUTINE TO HANDLE THE OPERATOR IS CALLED
                                                             */
/*************************************************************/

   IF  OPER ='+' THEN CALL ADD_IT(PTR_1,PTR_2,ANS_PTR,
                                  CAR_PTR,FLAG);
   IF  OPER ='-' THEN CALL SUB_IT(PTR_1,PTR_2,ANS_PTR,
                                  CAR_PTR,FLAG);
   IF  OPER ='*' THEN CALL MUL_IT(PTR_1,PTR_2,ANS_PTR,
                                  CAR_PTR,FLAG);
   IF  OPER ='/' THEN CALL DIV_IT(PTR_1,PTR_2,ANS_PTR,
                                  CAR_PTR,FLAG);
   IF ¬FLAG THEN FLAG = '1'B;
   GO TO DU;
   F_SCAN: PROC(CH) CHAR(1);
/*************************************************************/
/*
      F_SCAN SEARCHES WORK_S.FOR THE FIRST OPERATOR
   ENCOUNTERED AND RETURNS THIS OPERATOR. IT ALSO SETS POSIT
   TO THE LOCATION OF THE OPERATOR IN THE STRING.
                                                             */
/*************************************************************/
      DCL CH CHAR(240) VARYING;
      DCL CHARC CHAR(1);
      DO  WHILE(CHARC ¬=';');
          POSIT = POSIT + 1;
          CHARC = SUBSTR(CH,POSIT,1);
          IF CHARC <'#' | CHARC = '='
          THEN RETURN(CHARC);
      END;
      RETURN(CHARC);
   END F_SCAN;
   R_SCAN: PROC(X,Y);
/*************************************************************/
/*
      R_SCAN BEGINS AT POSIT AND SEARCHES WORK_S BACKWARDS
   GETTING THE FIRST TWO OPERANDS ENCOUNTERED. AFTER THE
   SECOND OPERAND HAS BEEN DETERMINED R_SCAN INSERTS A TEM-
   PORARY OPERAND INTO WORK_S WHICH INDICATED WHERE THE
   PARTIAL ANSWER OBTAINED FROM THE OPERANDS WILL BE LOCATED
                                                             */
/*************************************************************/
      DCL (X,Y) CHAR(3) VARYING;
      DCL (HOLD) CHAR(1);
      HOLD = ' ';
      PLACE = POSIT;
      DO WHILE(HOLD ¬= '#' & HOLD ¬= '@');
         PLACE = PLACE -1;
         HOLD = SUBSTR(PROB_S,PLACE,1);
      END;
      Y = SUBSTR(PROB_S,PLACE,POSIT-PLACE);
      PROB_S = SUBSTR(PROB_S,1,PLACE-1)||SUBSTR(PROB_S,
                                                POSIT+1);
      POSIT = PLACE;
      HOLD = ' ';
      DO WHILE(HOLD ¬= '#' & HOLD ¬='@');
         PLACE = PLACE-1;
         HOLD = SUBSTR(PROB_S,PLACE,1);
      END;
      X = SUBSTR(PROB_S,PLACE,POSIT-PLACE);
      IF PLACE = 1
      THEN PROB_S ='@'||TEMP_C||SUBSTR(PROB_S,POSIT);
      ELSE PROB_S = SUBSTR(PROB_S,1,PLACE-1)||'@'||TEMP_C||
                    SUBSTR(PROB_S,POSIT);
      POSIT = POSIT - 1;
      RETURN;
   END R_SCAN;
FINIS: END PROBSOL;
```

```
ADD_IT: PROC(X,Y,U,V,FLAG);

/*****************************************************************/
/*
      ADD_IT ADDS TWO NUMBERS IN THE MANNER THAT A STUDENT
   WOULD. THE TWO CORRESPONDING DIGITS IN EACH NUMBER ARE
   ADDED TOGETHER. IF THE RESULT IS GREATER THAN 9 THE
   CORRESPONDING DIGIT IN THE ANSWER IS THE SUM MODULUS 10.
   AND THE CARRY STORAGE FOR THE NEXT HIGHER DIGET POSITION
   IS SET TO THE SUM/10.
                                                              */
/*****************************************************************/


   DCL (X,Y,U,V,Z) PTR;
   DCL (DIG_1,DIG_2,GR_DIG) FIXED BIN(15);
   OCL FLAG BIT(1);
   ON ERROR GO TO CONT;
   DIG_1 = X->DIGETS; DIG_2 = Y->DIGETS;
   GR_DIG = MAX(DIG_1,DIG_2); LST_DIG = MIN(DIG_1,DIG_2);
   IF FLAG
   THEN DO;
           FREE U->PROBLEM;
           FREE V->PROBLEM;
        END;
CONT:SIZE = GR_DIG + 1;
   ALLOCATE PROBLEM;
   U = PP;
   ALLOCATE PROBLEM;
   V = PP;
   U->PROB = 0;
   V->PROB = 0;
   DO I = 1 TO LST_DIG;
      U->PROB(I) = X->PROB(I) + Y->PROB(I) + V-> PROB(I);
      IF U->PROB(I) >= 10
      THEN DO;
              V->PROB(I+1)= U->PROB(I)/10;
              U->PROB(I)= MOD(U->PROB(I),10);
           END;
   END;
   IF DIG_1 > DIG_2 THEN Z = X;
                    ELSE Z = Y;
   DO  I = LST_DIG +1 TO GR_DIG;
      U-> PROB(I) = Z->PROB(I) + V->PROB(I);
      IF U ->PROB(I)> 10
      THEN DO;
              V->PROB(I +1)= U->PROB(I)/10;
              U->PROB(I) = MOD(U->PROB(I),10);
           END;
   U->PROB(GR_DIG +1) = V->PROB(GR_DIG +1);
   RETURN;
END ADD_IT;
```

```
SUB_IT: PROC(X,Y,U,V,FLAG);

/*************************************************************/
/*
      SUB_IT SUBTRACTS TWO NUMBERS A DIGIT AT A TIME. IF
   THE DIGIT BEING SUBTRACTED IS GREATER THAN THE DIGIT FROM
   WHICH IT IS BEING SUBTRACTED THAN A BORROW IS PERFORMED
   FROM THE NEXT HIGHEST DIGIT AND THE CARRY DIGIT FOR THE
   DIGITS BEING SUBTRACTED IS INCREASED BY 10. THUS THE
   ANSWER DIGIT CORRESPONDING TO THE DIGITS BEING SUBTRACTED
   IS ALWAYS POSITIVE.
                                                            */
/*************************************************************/


   DCL (X,Y,U,V,Z) PTR;
   DCL (DIG_1,DIG_2,GR_DIG) FIXED BIN(15);
   DCL FLAG BIT(1);
   ON ERROR GO TO CONT;
   DIG_1 = X->DIGETS; DIG_2 = Y->DIGETS;
   GR_DIG = MAX(DIG_1,DIG_2); LST_DIG = MIN(DIG_1,DIG_2);
   IF FLAG
   THEN DO;
           FREE U->PROBLEM;
           FREE V->PROBLEM;
        END;
CONT:SIZE = GR_DIG;
   ALLOCATE PROBLEM;
   U = PP;
   ALLOCATE PROBLEM;
   V = PP;
   U->PROB = 0;
   V->PROB = 0;
   DO I = 1 TO LST_DIG;
      IF X->PROB(I) < Y->PROB(I)
      THEN DO;
              V->PROB(I) = 10;
              X->PROB(I + 1) = X->PROB(I+1) -1;
           END;
      U->PROB(I) = V->PROB(I)+ X->PROB(I) - Y->PROB(I);
   END;
   DO I = LST_DIG +1 TO GR_DIG;
       U->PROB(I) = X->PROB(I);
   END;
   RETURN;
END SUB_IT;
```

```
                MUL_IT:PROC(X,Y,U,V,FLAG);

 /*************************************************************/
 /*
       MULT_IT MULTIPLIES THE DIGITS OF THE MULTIPLICAND
   BY ONE DIGIT OF THE MULTIPLIER. CARRIES ARE PRODUCED WHEN
   THE RESULT OF THIS MULTIPLICATION EXCEEDS 9. ADD_IT IS
   THEN CALLED TO SUM THE ANSWER AND THE CARRIES PRODUCING
   A SUBMULTIPLE WHICH IS STORED IN'PART'. WHEN ALL OF THE
   DIGITS OF THE MULTIPLIER HAVE BEEN USED THE SUBMULTIPLES
   ARE TOTALED TO PRODUCE THE FINAL RESULT.
                                                          */
 /*************************************************************/

   DCL PART(*) CONTROLLED FIXED BIN(15);
   DCL (X,Y,U,V,Z,TU,TV) PTR;
   DCL (DIG_1,DIG_2,GR_DIG,TOTAL) FIXED BIN(15);
   DCL FLAG BIT(1);
   ON ERROR GO TO CONT;
   DIG_1 = X->DIGETS; DIG_2 = Y->DIGETS;
   GR_DIG = MAX(DIG_1,DIG_2); LST_DIG = MIN(DIG_1,DIG_2);
   IF FLAG
   THEN DO;
           FREE U->PROBLEM;
           FREE V->PROBLEM;
    END;
CONT:SIZE = DIG_1 + DIG_2;
   ALLOCATE PROBLEM; U = PP;
   ALLOCATE PROBLEM; V = PP;
   U->PROB = 0; V->PROB = 0;
   K= -1; FLAG = '0'B;
   ALLOCATE PART(DIG_2);
   PART = 0;
   DO I =1 TO DIG_2;
      K = K+1;
      DO J = 1 TO DIG_1;
         U->PROB(K+J) = Y-> PROB(I) * X->PROB(J);
         IF U->PROB(K+J) >10
         THEN DO;
                 V->PROB(K+J+1) = V->PROB(K+J+1)+U->PROB
                                                  (K+J)/10;
                 U->PROB(K+J)= MOD(U->PROB(K+J),10);
                 IF V->PROB(K+J+1) > 10
                 THEN DO;
                        V->PROB(K+J+2)=V->PROB(K+J+2) +
                                       V->PROB(K+J+1)/10;
                        V->PROB(K+J+1)=MOD(V->PROB(K+J+1),10
                      END;
              END;
      END;
      CALL ADD_IT(U,V,TU,TV,FLAG);
      MM=1;
      DO L = 1 TO TU->DIGETS-1;
          PART(K+1) = TU->PROB(L)*MM + PART(K+1);
          MM = MM * 10;
      END;
      FREE TV -> PROBLEM;
      FREE TU-> PROBLEM;
      U->PROB = 0; V->PROB = 0;
   END;
   TOTAL = 0;
   DO LL = 1 TO DIG_2;
       TOTAL = TOTAL + PART(LL);
   END;
   DO LLL= 1 TO U->DIGETS;
       U -> PROB(LLL) = MOD(TOTAL,10);
       TOTAL = TOTAL / 10;
   END;
   FLAG = '1'B;
   RETURN;
END MUL_IT;
```

```
DIV_IT:PROC(X,Y,U,V,FLAG);

/*************************************************************/
/*
      DIV_IT EXAMINES THE DIVIDEND STARTING AT THE HIGHEST
   PLACE VALUE AN LOCATES THE DIGIT WHERE THE DIVISOR WILL
   DIVIDE THE DIVIDEND. THE DIVISION IS PERFORMED AND THE
   REMAINDER IS ADDED TO THE PART OF THE DIVIDEND TO THE
   RIGHT OF THE DIGIT WHERE THE DIVISION OCCURRED AND THE
   PROCESS REPEATS. WHEN THE DIVISION IS COMPLETED THE UNITS
   DIGITS OF THE CARRY ARRAY CONTAINS THE REMAINDER.
                                                          */
/*************************************************************/


   DCL (X,Y,U,V,Z) PTR;
   DCL (DIG_1,DIG_2,GR_DIG) FIXED BIN(15);
   DCL FLAG BIT(1);
   DCL MULT FIXED BIN(15);
   DCL(DIVER,DIV) FIXED BIN(15);
   ON ERROR GO TO CONT;
   DIG_1 = X->DIGETS; DIG_2 = Y->DIGETS;
   GR_DIG = MAX(DIG_1,DIG_2); LST_DIG = MIN(DIG_1,DIG_2);
   IF FLAG
   THEN DO;
            FREE V->PROBLEM;
            FREE U->PROBLEM;
   END;
            SIZE = GR_DIG;
CONT:ALLOCATE PROBLEM;
   U = PP;
   ALLOCATE PROBLEM;
   V = PP;
   U->PROB = 0;
   V->PROB = 0;
   MULT = 1;
   DO  I = 1 TO DIG_2;
       DIV = DIV + (Y->PROB(I) * MULT);
       MULT = MULT * 10;
   END;
   MULT = 10;
   DIVER = 0;
   I = DIG_1;
START: DIVER =(X->PROB(I)*MULT/10) + DIVER * MULT;
   NUM = DIVER/DIV;
   IF NUM = 0
   THEN DO;
            I = I-1;
            IF I = 0
            THEN DO;
                     V->PROB(1) = DIVER;
                     GO TO FINISH;
                 END;
            GO TO START;
       END;
   ELSE DO;
            U->PROB(I)= DIVER/DIV;
            V->PROB(I)= MOD(DIVER,DIV);
            DIVER = V->PROB(I);
            I = I-1;
            IF I = 0 THEN GO TO FINISH;
            GO TO START;
       END;
FINISH:RETURN;
END DIV_IT;
```

```
ESTAB: PROC;

/*************************************************************/
/*
     ESTAB ESTABLISHES THE LEVELS, PROBLEM TYPES AND
  ASSOCIATED TREES FROM THE FILE'TREE'. THUS IT IS PART OF
  THE OFF-LINE I/O PACKAGE REQUIRED FOR COMMUNICATION WITH
  THE TEACHER.
                                                            */
/*************************************************************/

  DCL J_CHAR CHAR (9);
  DCL J_C CHAR(1);
  DCL YES FIXED BIN(15);
  DCL DEPTH FIXED BIN(15);
  DCL SET_PTR PTR;
  OPEN FILE(TREE) INPUT;
/*************************************************************/
/*
     THE CODE FROM 'GET_L' TO 'GET_P' GETS THE INFORMATION
  REQUIRED FOR ONE SET OF THE DATA STRUCTURE 'LEVELS'. WHEN
  THIS IS ACCOMPLISHED, THE INFORMATION FOR A 'PROBLEM
  TYPE' ASSOCIATED WITH THAT 'LEVEL' IS OBTAINED.
                                                            */
/*************************************************************/
GET_L: GET FILE(TREE) EDIT (BUFFER)(COL(1),A(80));
  IF SUBSTR(BUFFER,1,3) = 'EOF'
  THEN DO;
          DO I = LEV_NR+1 TO 10;
             LEV_PTR(I) = NULL;
          END;
          RETURN;
     END;
  LEVEL_C = SUBSTR(BUFFER,8,3);
  LEV_NR = LEVEL_C;
  NAME(LEV_NR) = SUBSTR(BUFFER,12,60);
  N_S(LEV_NR) = SUBSTR(BUFFER,73);
  GET FILE(TREE) EDIT(BUFFER)(COL(1),A(80));
  NR_P_A(LEV_NR) = SUBSTR(BUFFER,1,20);
  I = 0;
  N_P(LEV_NR) = 0;
/*************************************************************/
/*
     THE CODE TO 'BRANCHES' GETS THE INFORMATION FOR
  ESTABLISHING THE STRUCTURE 'PROB_TYPE'. WHEN THIS
  INFORMATION HAS BEEN OBTAINED, THE 'TREE' FOR THE PROBLEM
  TYPE IS ESTABLISHED BEFORE THE NEXT PROBLEM TYPE IS
  OBTAINED. WHEN ALL PROBLEM TYPES HAVE BEEN ESTABLISHED,
  CONTROL RETURNS TO 'GET_L' TO GET THE NEXT LEVEL.
                                                            */
/*************************************************************/
GET_P:GET FILE(TREE) EDIT(BUFFER)(COL(1),A(80));
  IF SUBSTR(BUFFER,1,3) = 'EOL'
  THEN GO TO SETUP;
  N_P(LEV_NR) = N_P(LEV_NR) + 1;
  I = I + 1;
  ALLOCATE PROB_TYPE;
  IF I = 1
  THEN DO;
          LEV_PTR(LEV_NR) = PTP;
     END;
  ELSE P_PTR->P_NEXT = PTP;
  ACTIVE = PTP;
  P_PTR = ACTIVE;
  ACTIVE -> P_NEXT = NULL;
  NR_ST_C = SUBSTR(BUFFER,1,15);
  NR_ST = NR_ST_C;
  ACTIVE -> N_O_S = NR_ST;
  PROB_S = '';
```

```
/*****************************************************/
/*
      THE PROBLEM FORMAT IS OBTAINED AND CONVERTED TO
   POLISH.
                                                      */
/*****************************************************/
AGAIN: GET FILE(TREE) EDIT(BUFFER)(COL(1),A(80));
   SEMI = INDEX(BUFFER,';');
   IF  SEMI = 0
   THEN DO;
           PROB_S = PROB_S || BUFFER;
           GO TO AGAIN;
        END;
   PROB_S = PROB_S || SUBSTR(BUFFER,1,SEMI);
   SIZE = LENGTH(PROB_S);
   ALLOCATE PROBLEM_S;
   ACTIVE -> PROB_I = PSP;
   N_ACTIVE = PSP;
   N_ACTIVE->PROB_STR = PROB_S;
   CALL POLISH(PROB_S);
   SIZE = LENGTH(PROB_S);
   ALLOCATE PROBLEM_S;
   ACTIVE->PROB_P = PSP;
   N_ACTIVE = PSP;
   N_ACTIVE ->PROB_STR = PROB_S;
   J = 0;
/*****************************************************/
/*
      THE CONDITIONS FOR HARD, MEDIUM AND EASY PROBLEMS ARE
   OBTAINED.
                                                      */
/*****************************************************/
P2:COND_S ='';
AGAIN1:
   GET FILE(TREE) EDIT(BUFFER)(COL(1),A(80));
   SEMI = INDEX(BUFFER,';');
   IF SEMI = 0
   THEN DO;
           COND_S = COND_S || BUFFER;
           GO TO AGAIN;
        END;
   COND_S = COND_S || SUBSTR(BUFFER,1,SEMI);
   SIZE = LENGTH(COND_S);
   ALLOCATE CONDITIONS;
   N_ACTIVE = CP;
   N_ACTIVE->COND = COND_S;
   J= J + 1;
   IF  J = 1
   THEN DO;
           ACTIVE -> HARD_P = CP;
           GO TO P2;
        END;
   IF J = 2
   THEN DO;
           ACTIVE -> MED_P = CP;
           GO TO P2;
        END;
   IF J = 3
   THEN ACTIVE-> EASY_P = CP;
   DEPTH = 0;
/*****************************************************/
/*
      THE TREES FOR EACH PROBLEM TYPE ARE OBTAINED. THE
   CONDITIONS AND PROCESS FOR EACH NODE AT THE SAME LEVEL
   ARE OBTAINED FIRST. THEN THE FIRST NODE AT THAT LEVEL IS
   EXPANDED TO THE NEXT LEVEL BY OBTAINING ALL CONDITIONS
   AND PROCESSES AT THE NEXT LEVEL ASSOCIATED WITH THE NODE
   BEING EXPANDED. THEN THE PROCESS REPEATS. THUS THE NODES
   ARE ESTABLISHED IN A BREADTH-WISE MANNER AT EACH LEVEL,
   BUT EACH LEVEL IS EXPANDED IN A DEPTH-WISE MANNER.
                                                      */
/*****************************************************/
```

```
BRANCHES:
   DEPTH= DEPTH + 1;
   K = 0;
P3:GET FILE(TREE) EDIT(BUFFER)(COL(1),A(80));
   ALLOCATE NODE;
 /*****************************************************/
 /*
      'EOC' INDICATES THAT ALL CONDITIONS AND PROCESSES AT
   THAT LEVEL HAVE BEEN OBTAINED AND PROCEED TO THE NEXT
   NODE.
                                                       */
 /*****************************************************/
   IF  SUBSTR(BUFFER,1,3) = 'EOC'
   THEN DO;
           N_ACTIVE = ACTIVE-> RET_P;
           IF DEPTH = 1
           THEN ACTIVE = N_ACTIVE -> P_BRANCH;
           ELSE ACTIVE = N_ACTIVE -> N_BRANCH;
    AGAIN4:N_ACTIVE = ACTIVE -> PROC_P;
 /*****************************************************/
 /*
      IF THE PROCESS AT THE NODE IS 'HA:;' THEN THE NODE
   HAS NO LOWER LEVELS TO EXPAND. GO TO NEXT NODE AT THE
   SAME LEVEL.
                                                       */
 /*****************************************************/
           IF N_ACTIVE -> COND ¬= 'HA:;'
           THEN GO TO BRANCHES;
           ELSE DO;
                   ACTIVE -> N_BRANCH = NULL;
 /*****************************************************/
 /*
      IF N_NEXT IS NULL, THEN THERE ARE NO MORE NODES AT
   THAT LEVEL TO BE EXPANDED. RETURN TO NEXT HIGHER LEVEL
   AND EXPAND THE NEXT NODE AT THAT LEVEL.
                                                       */
 /*****************************************************/
            AGAIN5:IF ACTIVE -> N_NEXT ¬=NULL
                   THEN DO;
                           ACTIVE = ACTIVE -> N_NEXT;
                           GO TO AGAIN4;
                        END;
                   ELSE DO;
                           DEPTH = DEPTH -1;
                           IF DEPTH = 0
                           THEN GO TO GET_P;
                           ACTIVE = ACTIVE -> RET_P;
                           GO TO AGAIN5;
                        END;
                END;
      END;
   K = K+1;
   IF DEPTH = 1
   THEN DO;
           IF K = 1
           THEN ACTIVE -> P_BRANCH = NP;
           ELSE ACTIVE-> N_NEXT = NP;
      END;
   ELSE DO;
           IF K = 1
           THEN ACTIVE -> N_BRANCH = NP;
           ELSE ACTIVE -> N_NEXT  = NP;
      END;
   N_ACTIVE = NP;
   N_ACTIVE -> N_NEXT = NULL;
   IF K = 1
   THEN N_ACTIVE -> RET_P = ACTIVE;
   ELSE N_ACTIVE -> RET_P = ACTIVE -> RET_P;
   ACTIVE = N_ACTIVE;
   NR_ST_C = SUBSTR(BUFFER,1,15);
   NR_ST = NR_ST_C;
   ACTIVE -> SUCCESS = NR_ST;
```

```
   COND_S = '';
AGAIN2:
 /*****************************************************/
 /*
      GET THE CONDITION LIST FOR DETERMINING IF A NODE IS
   APPLICABLE.
                                                       */
 /*****************************************************/
   GET FILE(TREE) EDIT(BUFFER)(COL(1),A(80));
   SEMI = INDEX(BUFFER,';');
   IF SEMI = 0
   THEN DO;
           COND_S = COND_S||BUFFER;
           GO TO AGAIN2;
        END;
   COND_S = COND_S||SUBSTR(BUFFER,1,SEMI);
   SIZE = LENGTH(COND_S);
   ALLOCATE CONDITIONS;
   ACTIVE -> COND_P = CP;
   N_ACTIVE = CP;
   N_ACTIVE -> COND = COND_S;
   COND_S = '';
AGAIN3:
 /*****************************************************/
 /*
      GET THE PROCESS TO BE DONE WHEN THE NODE IS
   APPLICABLE.
                                                       */
 /*****************************************************/
   GET FILE(TREE) EDIT(BUFFER)(COL(1),A(80));
   SEMI = INDEX(BUFFER,';');
   IF SEMI = 0
   THEN DO;
           COND_S = COND_S || BUFFER;
           GO TO AGAIN3;
        END;
   COND_S = COND_S || SUBSTR(BUFFER,1,SEMI);
   SIZE = LENGTH(COND_S);
   ALLOCATE CONDITIONS;
   ACTIVE-> PROC_P = CP;
   N_ACTIVE = CP;
   N_ACTIVE -> COND = COND_S;
   GO TO P3;
SETUP:
 /*****************************************************/
 /*
      DETERMINE THE ARGUMENTS USED IN THE PROBLEM FORMAT
   AND CREATE THE ARGUMENT LIST.
                                                       */
 /*****************************************************/
   DO L = 1 TO N_P(LEV_NR);
      IF L = 1
      THEN ACTIVE = LEV_PTR(LEV_NR);
      ELSE ACTIVE = ACTIVE -> P_NEXT;
      N_ACTIVE = ACTIVE -> PROB_P;
      SIZE = 0;
      DO J = 1 TO 26;
         J_CHAR = J;
         IF J < 10
         THEN J_C = SUBSTR(J_CHAR,9,1);
         ELSE J_C = SUBSTR(J_CHAR,8,2);
         YES = INDEX(N_ACTIVE->PROB_STR,J_C);
         IF YES ¬= 0
         THEN SIZE = SIZE + 1;
      END;
      ALLOCATE ARGUEMENTS;
      ACTIVE -> ARG_P = AP;
      SET_PTR = AP;
      K = 0;
      DO J = 1 TO 26;
         J_CHAR = J;
         IF J < 10
```

```
            THEN J_C = SUBSTR(J_CHAR,9,1);
            ELSE J_C = SUBSTR(J_CHAR,8,2);
            YES = INDEX(N_ACTIVE->PROB_STR,J_C);
            IF YES ¬= 0
            THEN DO;
                    K=K+1;
                    SET_PTR-> ARGUE(K) = TRANS(J);
                 END;
         END;
      END;
      GO TO GET_L;
END ESTAB;



                        REFILE: PROC;

 /*********************************************************/
 /*
       REFILE IS USED TO UPDATE THE FILE  TREE  BY RESTORING
    THE LEVELS,PROBLEM TYPES,AND TREES TO BACK-UP STORAGE,
    WITH ANY CORRECTIONS THAT MAY HAVE BEEN MADE. IT CREATES
    THE FILE  COPTREE IN THE SAME MANNER  ESTAB  WOULD EXPECT
    TO FIND THE DATA. THEN THE FILE  TREE  MAY BE ERASED AND
    THE FILE  COPTREE  INSERTED IN ITS PLACE.
                                                          */
 /*********************************************************/

    DCL (N_S_CH,N_O_S_C,SUC_C) CHAR(15);
    DCL N_S_C  CHAR(7);
    DCL LEVEL_CH CHAR(9);
    DCL LEVEL_C CHAR(3);
    DCL(DEPTH,TIMES) FIXED BIN(15);
    DCL (P_PTR,LOC_PTR) PTR;
    DCL TOP BIT(1);
    OPEN FILE(COPTREE) OUTPUT ;
    DO I = 1 TO 10 WHILE(LEV_PTR(I) ¬= NULL);
 /*********************************************************/
 /*
       GET DATA FOR LEVEL AND MAKE UP LEVEL CARDS.
                                                          */
 /*********************************************************/
      N_S_CH = N_S(I);
      N_S_C =  SUBSTR(N_S_CH,9);
      LEVEL_CH = I;
      LEVEL_C = SUBSTR(LEVEL_CH,7);
      BUFFER = 'LEVEL '||LEVEL_C||' '||NAME(I)||' '||N_S_C;
      PUT FILE(COPTREE) EDIT(BUFFER)(COL(1),A(80));
      J = 0;
 /*********************************************************/
 /*
       THE NEXT SECTION OF CODE GETS THE INFORMATION
    REGARDING A PROBLEM TYPE .
                                                          */
 /*********************************************************/
NEXT_P: J = J+ 1 ;
      IF J = 1
      THEN P_PTR  = LEV_PTR(I);
      ELSE P_PTR = P_PTR -> P_NEXT; ACTIVE  = P_PTR;
 /*********************************************************/
 /*
       THERE ARE NO MORE PROBLEMS IN THE LEVEL.
                                                          */
 /*********************************************************/
      IF ACTIVE = NULL
      THEN DO;
              BUFFER = 'EOL';
              PUT FILE(COPTREE) EDIT(BUFFER)(COL(1),A(80));
              GO TO ND;
           END;
```

```
/*****************************************************/
/*
     GET THE NUMBER OF PROBLEMS ASKED BY THE THE COMPUTER
  OF THIS PROBLEM TYPE.
                                                      */
/*****************************************************/
     N_O_S_C = ACTIVE->N_O_S;
     BUFFER  = N_O_S_C;
     PUT FILE(COPTREE) EDIT(BUFFER)(COL(1),A(80));
/*****************************************************/
/*
     GET THE PROBLEMS FORMAT.
                                                      */
/*****************************************************/
     N_ACTIVE = ACTIVE -> PROB_I;
     TIMES = N_ACTIVE -> P_NR / 80;
     PROB_S = N_ACTIVE -> PROB_STR;
     DO K = 1 TO TIMES;
        BUFFER = SUBSTR(PROB_S,1,80);
        PUT FILE(COPTREE) EDIT(BUFFER)(COL(1),A(80));
        PROB_S = SUBSTR(PROB_S,81);
     END;
     IF PROB_S ¬= ''
     THEN DO;
             BUFFER = PROB_S;
             PUT FILE(COPTREE) EDIT(BUFFER)(COL(1),A(80));
          END;
/*****************************************************/
/*
     GET THE CONDITIONS FOR HARD,MEDIUM AND EASY PROBLEMS.
                                                      */
/*****************************************************/
    DO K = 1 TO 3;
       IF K = 1
       THEN N_ACTIVE = ACTIVE -> HARD_P;
       IF K = 2
       THEN N_ACTIVE = ACTIVE -> MED_P;
       IF K = 3
       THEN N_ACTIVE = ACTIVE -> EASY_P;
       TIMES = N_ACTIVE -> C_NR/80;
       COND_S = N_ACTIVE -> COND;
       DO L = 1 TO TIMES;
          BUFFER = SUBSTR(COND_S,1,80);
          PUT FILE(COPTREE) EDIT(BUFFER)(COL(1),A(80));
          COND_S = SUBSTR(COND_S,81);
       END;
       IF COND_S ¬= ''
       THEN DO;
               BUFFER = COND_S;
               PUT FILE(COPTREE) EDIT(BUFFER)(COL(1),A(80)
            END;
     END;
/*****************************************************/
/*
     GET THE ASSISTANCE TREE ASSOCIATED WITH THE PROBLEM
  TYPE JUST PLACED IN THE FILE  COPTREE.
                                                      */
/*****************************************************/
     DEPTH = 0;
     ACTIVE = ACTIVE -> P_BRANCH;
     TOP = '1'B;
NEXT_C:
     DEPTH = DEPTH + 1;  LOC_PTR = ACTIVE;
     DO WHILE(LOC_PTR ¬= NULL);  ACTIVE  = LOC_PTR;
        SUC_C = ACTIVE -> SUCCESS;
        BUFFER = SUC_C;
        PUT FILE(COPTREE) EDIT(BUFFER)(COL(1),A(80));
```

```
    CON:
/*****************************************************/
/*
      GET THE CONDITIONS ON A BRANCH.
                                                      */
/*****************************************************/
        LOC_PTR = ACTIVE -> COND_P;
        COND_S = LOC_PTR -> COND;
        TIMES = LOC_PTR -> C_NR/80;
        DO K = 1 TO TIMES;
           BUFFER = SUBSTR(COND_S,1,80);
           PUT FILE(COPTREE) EDIT(BUFFER)(COL(1),A(80));
           COND_S = SUBSTR(COND_S,81);
        END;
        IF COND_S ¬= ''
        THEN DO;
                BUFFER = COND_S;
                PUT FILE(COPTREE) EDIT(BUFFER)(COL(1),A(80))
             END;
    PRO:
/*****************************************************/
/*
      GET THE PROCEDURE TO FOLLOW IN THIS BRANCH.
                                                      */
/*****************************************************/
        LOC_PTR = ACTIVE -> PROC_P;
        COND_S = LOC_PTR -> COND;
        TIMES = LOC_PTR-> C_NR/80;
        DO K = 1 TO TIMES;
           BUFFER = SUBSTR(COND_S,1,80);
           PUT FILE(COPTREE) EDIT(BUFFER)(COL(1),A(80));
           COND_S = SUBSTR(COND_S,81);
       END;
        IF COND_S ¬= ''
        THEN DO;
                BUFFER = COND_S;
                PUT FILE(COPTREE) EDIT(BUFFER)(COL(1),A(80))
             END;
        LOC_PTR = ACTIVE -> N_NEXT;
     END;
/*****************************************************/
/*
      END OF NODES AT A GIVEN LEVEL ASSOCIATED WITH ONE
  NODE AT THE NEXT HIGHER LEVEL. DECIDE WHICH NODE TO GO TO
  NEXT.
                                                      */
/*****************************************************/
     BUFFER = 'EOC';
     PUT FILE(COPTREE) EDIT(BUFFER)(COL(1),A(80));
  P1:IF TOP
     THEN DO;
             N_ACTIVE = ACTIVE -> RET_P;
             ACTIVE = N_ACTIVE -> P_BRANCH;
             N_ACTIVE = ACTIVE;
             ACTIVE = N_ACTIVE -> N_BRANCH;
             TOP = '0'B;
          END;
     ELSE DO;
             N_ACTIVE = ACTIVE -> RET_P;
             ACTIVE = N_ACTIVE -> N_BRANCH;
             N_ACTIVE = ACTIVE;
             ACTIVE = N_ACTIVE -> N_BRANCH;
          END;


  P2:IF ACTIVE = NULL
     THEN DO;
             ACTIVE = N_ACTIVE -> N_NEXT;
          P3:IF ACTIVE = NULL
             THEN DO;
                     N_ACTIVE = N_ACTIVE -> RET_P;
```

```
                        DEPTH = DEPTH-1;
                        IF DEPTH = 0
                        THEN GO TO NEXT_P;
                        ACTIVE = N_ACTIVE -> N_NEXT;
                        GO TO P3;
                    END;
                ELSE DO;
                        N_ACTIVE = ACTIVE;
                        ACTIVE = N_ACTIVE -> N_BRANCH;
                        GO TO P2;
                    END;
            END;
        GO TO NEXT_C;
ND:END;
 /*****************************************************/
 /*
       ALL LEVELS HAVE BEEN RESTORED TO BACK-UP STORAGE.
                                                       */
 /*****************************************************/
   BUFFER ='EOF' ;
   PUT FILE(COPTREE) EDIT(BUFFER)(COL(1),A(80));
   RETURN;
END REFILE;
```

```
POLISH: PROC (WORK_S);


/*****************************************************************/
/*
     POLISH CONVERTS THE INFIX EXPRESSION OF THE PROBLEM
  INTO AN EQUIVALENT POLISH EXPRESSION OF THE PROBLEM. IT
  CALLS THE ROUTINE  PREC  TO DETERMINE THE RELATIVE
  HIERARCHY OF OPERATORS. AFTER THE STRING HAS BEEN CHANGED
  INTO POLISH THE ARGUMENTS ARE REPLACED BY '# NUMBER'
  WHERE THE NUMBER TELLS WHICH POINTER IN THE ARRAY
  ARG_PTRS  POINTS TO THE NUMBER GENERATED TO REPLACE THAT
  ARGUMENT.
     WORK_S - COPY OF THE INFIX EXPRESSION OF THE PROBLEM.
     STACK_A - STORES OPERATORS TILL TIME TO INSERT THEM
               INTO THE POLISH STRING.
     STACK_B - POLISH EXPRESSION OF THE PROBLEM
                                                              */
/*****************************************************************/


  DCL  WORK_S  CHAR(240) VARYING;
  DCL FUDGE CHAR(9) VARYING;
  DCL NR FIXED BIN(15);
  DCL OP1 CHAR(1) INITIAL(' ');
  DCL STACK_A CHAR(80) VARYING;
  DCL STACK_B CHAR(240) VARYING;
  DCL OP CHAR(1);
  DCL OP2 CHAR(3) VARYING;
  DCL OPNDS CHAR(26) INITIAL('ABCDEFGHIJKLMNOPQRSTUVWXYZ');
  DCL PREC  ENTRY(CHAR(1)) RETURNS(FIXED BIN(15,0));
  START:IF WORK_S = '' THEN GO TO REPLACE;
     ELSE DO;
/*****************************************************************/
/*
     GET THE FIRST NON-BLANK CHARACTER IN WORK_S.
                                                              */
/*****************************************************************/
             OP = SUBSTR(WORK_S,1,1);
             IF OP = ' '
             THEN DO;
                    WORK_S = SUBSTR(WORK_S,2);
                    GO TO START;
                  END;
/*****************************************************************/
/*
     IF THE CHARACTER IS NOT AN OPERAND THEN IT MUST BE AN
  OPERATOR. GO TO  PAREN AND CHECK FOR CLOSING PARENTHESES.
                                                              */
/*****************************************************************/
             IF INDEX(OPNDS,OP) = 0 THEN GO TO PARENQ;
             STACK_B = STACK_B || OP;
             WORK_S = SUBSTR(WORK_S,2);
             IF SUBSTR(WORK_S,1,1)=' ' THEN WORK_S=SUBSTR
                                              (WORK_S,2);
/*****************************************************************/
/*
     COMPARE OPERATOR IN WORK_S WITH OPERATOR ON STACK_A.
  IF LESS THAN,THEN REMOVE OPERATOR FROM STACK_A AND ADD TO
  STACK_B.
                                                              */
/*****************************************************************/
       LOOP: IF (WORK_S = '')|(STACK_A = '') |(PREC(SUBSTR
             (WORK_S,1,1))>PREC(SUBSTR(STACK_A,1,1)))
             THEN GO TO START;
             ELSE DO;
                    STACK_B = STACK_B||SUBSTR(STACK_A,1,1);
                    STACK_A = SUBSTR (STACK_A, 2);
                    GO TO LOOP;
                    END;
```

```
/*****************************************************************/
/*
      IF OPERATOR IS NOT A ')' THEN CHECK FOR END OF
  STRING.IF NOT END OF STRING THEN ADD OPERATOR TO STACK_A.
                                                              */
/*****************************************************************/
      PARENQ:IF SUBSTR (WORK_S, 1, 1) ¬= ')'
             THEN DO;
                    IF SUBSTR (WORK_S,1,1) = ';'
                    THEN DO;
                           STACK_B = STACK_B || SUBSTR
                                          (STACK_A,1,1);
                           STACK_A = SUBSTR (STACK_A,2);
                         END;
                    STACK_A = SUBSTR(WORK_S,1,1)||STACK_A;
                    WORK_S = SUBSTR (WORK_S,2);
                    GO TO START;
                  END;
             ELSE DO;
                    STACK_A = SUBSTR (STACK_A,2);
                    WORK_S = SUBSTR(WORK_S,2);
                    GO TO LOOP;
                  END;
             END;
/*****************************************************************/
/*
      REPLACE ALL ARGUMENTS BY '# NUMBER'.
                                                              */
/*****************************************************************/
  REPLACE: DO WHILE (OP ¬= '=');
             OP = SUBSTR (STACK_B, 1,1);
             STACK_B = SUBSTR(STACK_B,2);
             IF OP >= 'A' & OP <= 'Z'
             THEN DO;
                    NR = INDEX (OPNDS, OP);
                    FUDGE = NR;
                       OP1 = ' ';
                    DO WHILE(OP1 = ' ');
                       OP1 = SUBSTR(FUDGE,1,1);
                       IF OP1=' '
                       THEN FUDGE = SUBSTR(FUDGE,2);
                    END;
                    OP2 = '#' || FUDGE;
                    WORK_S = WORK_S || OP2;
                    GO TO BOTTOM;
                  END;
             WORK_S = WORK_S || OP;
      BOTTOM:END;
  RETURN;
  PREC: PROC(X) FIXED BIN(15,0) ;
      DCL X  CHAR(1);
      DCL H  CHAR(8) INITIAL ('*/+-=);(');
      DCL HNUM(8) FIXED BIN(15,0) INITIAL(3,3,2,2,2,1,1,0);
      RETURN(HNUM(INDEX(H,X)));
  END PREC;
END POLISH;
```

```
BIG_POL:PROC(COND_S);

/*****************************************************/
/*
      BIG_POL CONVERTS THE CONDITION LIST, SPECIFYING
   WHETHER A NODE IS APPLICABLE, INTO A POLISH  EXPRESSION
   OF THE SAME STRING AND PASSES THIS ON TO CON_INT FOR
   INTERPRETATION. THE CONDITION LIST IS PASSED TO BIG_POL
   AS A PARAMETER. THE ROUTINE HIER IS USED TO CHECK THE
   HIERACHY OF OPERATIONS. THE ROUTINES CH_RAT AND CONVERT,
   CONVERT THE DOUBLE-CHARACTER OPERATORS, '//', '**', '<=',
   '>=', INTO A SINGLE CHARACTER.
      COND_S - CONTAINS THE CHARACTER STRING
      STACK_A - STORES OPERATORS UNTIL THEY ARE TO BE
                INSERTED IN THE POLISH STRING
      STACK_B - CONTAINS THE POLISH STRING
      OPAND   - CONTAINS THE OPERAND TO BE REMOVED FROM
                COND_S AND ADDED TO STACK B
                                                     */
/*****************************************************/

DCL ANS CHAR(1);
   DCL TRUE BIT(1);
   DCL(STACK_A,STACK_B,COND_S) CHAR(80)VARYING;
   DCL HIER ENTRY (CHAR(1)) RETURNS(FIXED BIN(15,0));
   DCL OPAND CHAR(10) VARYING;
   STACK_A = '';
   STACK_B = '';
TOP:IF COND_S ='' THEN GO TO FINI ;
   ELSE DO;
           DO WHILE(SUBSTR(COND_S,1,1) =' ');
              COND_S = SUBSTR(COND_S,2);
           END;
/*****************************************************/
/*
      WHEN AN OPERATOR IS ENCOUNTERED, GO TO PAREN TO SEE
   IF IT IS A CLOSING PARENTHESES.
                                                     */
/*****************************************************/
           IF SUBSTR(COND_S,1,1) < 'A' THEN GO TO PAREN;
           OPAND = '';
/*****************************************************/
/*
      THIS SEGMENT OF CODING REMOVES THE OPERANDS FROM
   COND_S AND PLACES THEM IN STACK_B WITH A COMMA TO MARK
   THE END OF AN OPERAND.
                                                     */
/*****************************************************/
      CONT:DO WHILE (SUBSTR(COND_S ,1,1)>='A');
              OPAND = OPAND||SUBSTR(COND_S,1,1);
              COND_S = SUBSTR(COND_S,2);
           END;
           IF SUBSTR(COND_S,1,1) ='.'
           THEN DO;
                   OPAND = OPAND || '.';
                   COND_S = SUBSTR(COND_S,2);
                   GO TO CONT;
                END;
           STACK_B = STACK_B||OPAND||',';
     LOOP:DO WHILE(SUBSTR(COND_S,1,1) = ' ');
              COND_S = SUBSTR(COND_S,2);
           END;
           CALL CH_RAT;
/*****************************************************/
/*
      CHECK HIERARCHEY OF OPERATOR WITH OPERATOR ON TOP OF
   STACK_A. IF IT IS LESS THAN OPERATOR ON STACK_A IT IS
   MOVED TO STACK B AND LOOP TO CHECK NEXT OPERATOR ON
   STACK A. ELSE GO TO TOP AND START AGAIN.
                                                     */
/*****************************************************/
```

```
     LOOP1:IF (COND_S ='') | (STACK_A = '') |
               (HIER(SUBSTR(COND_S,1,1)) > HIER(SUBSTR
             (STACK_A,1,1))) THEN GO TO TOP;
           ELSE DO;
                   STACK_B = STACK_B||SUBSTR(STACK_A,1,1);
                   STACK_A = SUBSTR(STACK_A,2);
                   GO TO LOOP1;
                 END;
 /*************************************************************/
 /*
       IF OPERATOR IS NOT A CLOSING PARENTHESES, THEN ADD
   OPERATOR TO STACK_A AND START AGAIN. IF IT IS, REMOVE
   OPENING PARENTHESES FROM STACK A AND CHECK TO SEE IF NEXT
   OPERATOR IS A ¬. THEN GO TO LOOP TO CHECK OPERATOR
   FOLLOWING THE CLOSING PARENTHESES.
                                                             */
 /*************************************************************/
     PAREN:IF SUBSTR(COND_S,1,1) ¬=')'
           THEN DO;
                   CALL CH_RAT;
                   STACK_A = SUBSTR(COND_S,1,1) || STACK_A;
                   COND_S = SUBSTR(COND_S,2);
                   GO TO TOP;
                 END;
           ELSE DO;
                   COND_S = SUBSTR(COND_S,2);
                   STACK_A = SUBSTR(STACK_A,2);
                   IF SUBSTR(STACK_A,1,1) = '¬'
                   THEN DO;
                           STACK_B = STACK_B || SUBSTR
                                     (STACK_A,1,1);
                           STACK_A = SUBSTR(STACK_A,2);
                         END;
                   GO TO LOOP;
                 END;
       END;
   CH_RAT: PROC;
       DCL CONVERT ENTRY (CHAR(2)) RETURNS (CHAR(1));
       DCL OPTRS CHAR(2);
       DCL OPTR  CHAR(1);
       IF COND_S ='' THEN RETURN;
       IF SUBSTR(COND_S,1,1)='/' | SUBSTR(COND_S,1,1)='*' |
          SUBSTR(COND_S,1,1)='>'  | SUBSTR(COND_S,1,1)='<'
       THEN DO;
               IF SUBSTR(COND_S,2,1)='/' | SUBSTR(COND_S,2,1)
                 ='*' | SUBSTR(COND_S,2,1)='='
               THEN DO;
                       OPTRS = SUBSTR(COND_S,1,2);
                       COND_S = SUBSTR(COND_S,3);
                       OPTR = CONVERT(OPTRS);
                       COND_S = OPTR || COND_S;
                    END;
            END;
       RETURN;
       CONVERT: PROC(X) CHAR(1);
       DCL X CHAR(2) ;
       DCL COMPARE  CHAR(8) INITIAL('**//>=<=');
       DCL REPLACE  CHAR(8) INITIAL('$ ? % : ') ;
       RETURN (SUBSTR(REPLACE,INDEX(COMPARE,X),1));
       END CONVERT;
   END CH_RAT;
HIER:PROC(X) FIXED BIN(15);
   DCL X CHAR(1);
   DCL H CHAR(17) INITIAL('¬/*+-$?%:<>=&|);(');
   DCL HNUM(17) FIXED BIN(15)INITIAL(7,6,6,5,5,4,4,4,4,4,4,4,4
   RETURN (HNUM(INDEX(H,X)));
END HIER;
FINI:CALL CON_INT(STACK_B,TRUE);
END BIG_POL;
```

```
CON_INT:PROC (COND_S,TRUTH);

/*****************************************************/
/*
     CON_INT INTERPRETS THE LIST OF CONDITIONS, FOR THE
  APPLICABILITY OF NODES IN THE TREE, TO DETERMINE IF THE
  NODE SHOULD BE SELECTED  IT OPERATES ON THE POLISH
  EXPRESSION OF THESE CONDITIONS WHICH IS PASSED AS A
  PARAMETER, AND IT RETURNS THE TRUTH VALUE OF THE LIST OF
  CONDITIONS. THE MAIN VARIABLES USED BY CON_INT ARE:
     COND_S - WHICH CONTAINS THE POLISH EXPRESSION OF THE
              LIST OF CONDITIONS.
     STACK - WHICH IS A PUSHDOWN STACK CONTAINING THE
             YTRUTH VALUE OF CONDITIONS IN THE STRING OR
             THE LOCATION OF WHERE ELEMENTS OF CONDITIONS
             ARE STORED.
     TOP - WHICH IS THE NEXT ITEM TO BE ADDED TO THE STACK
     OP_STR -WHICH STORES ELEMENTS OF CONDITIONS IN THE
             LIST.
     TRUTH - THE TRUTH VALUE OF THE STRING.
                                                      */
/*****************************************************/


  DCL COND_WORD(10) CHAR(2);
  DCL TRUTH BIT(1);
  DCL COND_S CHAR(240) VARYING;
  DCL (TOP,OPND) CHAR(10) VARYING;
  DCL STACK CHAR(2) CONTROLLED;
  DCL OPTRS CHAR(14) INITIAL('&|¬>=<?%:$+-*/');
  DCL (OPTR,CHARC) CHAR(1);
  DCL NR_C CHAR(9) VARYING;
  DCL (FLAG,FLOG) BIT(1);
  DCL DIGET FIXED BIN(15);
  DCL PERIOD FIXED BIN(15);
  DCL (NR,NR_1,NR_2,OPTR_NR) FIXED BIN(15);
  DCL KEY CHAR(2);
  DCL (LAB(10),LAB_L(0:14),LBL(14)) LABEL;
  DCL OP_STR(10:25) FIXED BIN(15);
  NR = 10;  TOP = '';
GET: OPND = '';
  IF COND_S = ''
  THEN GO TO LAB_L(0);
/*****************************************************/
/*
     THIS SEGMENT OF CODING EXTRACTS THE OPERANDS OF THE
  CONDITIONS FROM COND_S.
                                                      */
/*****************************************************/
GOT:DO WHILE(SUBSTR(COND_S,1,1)>='A');
     CHARC = SUBSTR(COND_S,1,1);
     OPND = OPND || CHARC;
     COND_S = SUBSTR(COND_S,2);
  END;
  IF  SUBSTR(COND_S,1,1) ='.'
  THEN DO;
         OPND = OPND || '.';
         COND_S = SUBSTR(COND_S,2);
         GO TO GOT;
       END;
  OPTR = SUBSTR(COND_S,1,1);
  COND_S = SUBSTR(COND_S,2);
/*****************************************************/
/*
     A COMMA INDICATES THE END OF A KEY WORD AND A CHECK
  IS MADE TO SEE IF A TRUTH VALUE CAN BE ASSIGNED TO THAT
  KEY WORD.
                                                      */
/*****************************************************/
```

```
   IF  OPTR = ','
   THEN DO;
            IF SUBSTR(OPND,1,1) <='Z'
            THEN DO;
                     KEY = SUBSTR(OPND,1,2);
                     DO I = 1 TO 10;
                        IF KEY = COND_WORD(I)
                        THEN GO TO LAB(I);
                     END;
             LAB(1):IF S_ANS = M_ANS
                     THEN OPND = '1 ';
                     ELSE OPND = '0 ';
                     GO TO CONT;
             LAB(2):IF S_ANS ¬= M_ANS
                     THEN OPND = '1 ';
                     ELSE OPND = '0 ';
                     GO TO CONT;
             LAB(3):IF S_RATE = 2
                     THEN OPND = '1 ';
                     ELSE OPND = '0 ';
                     GO TO CONT;
             LAB(4):IF S_RATE = 1
                     THEN OPND = '1 ';
                     ELSE OPND = '0 ';
                     GO TO CONT;
             LAB(5):IF S_RATE = 0
                     THEN OPND = '1 ';
                     ELSE OPND = '0 ';
                     GO TO CONT;
             LAB(6):IF PROB_DIF = 3
                     THEN OPND = '1 ';
                     ELSE OPND = '0 ';
                     GO TO CONT;
             LAB(7):IF PROB_DIF = 2
                     THEN OPND = '1 ';
                     ELSE OPND = '0 ';
                     GO TO CONT;
             LAB(8):IF PROB_DIF = 1
                     THEN OPND = '1 ';
                     ELSE OPND = '0 ';
                     GO TO CONT;
/*************************************************************/
/*
      IF  THE KEY WORD, SANS OR MANS, IS ENCOUNTERED, THEN
  IT IS CHECKED TO SEE IF IT IS QUALIFIED. THE VALUE
  INDICATED BY THIS OPTIONALLY QUALIFIED KEY WORD IS PLACED
  IN OP_STR AND THE VALUE OF OPND IS CHANGED TO THE
  LOCATION IN OP_STR WHERE THE VALUE IS LOCATED.
                                                          */
/*************************************************************/
             LAB(9):PERIOD = INDEX(OPND,'.');
                     FLAG ='0'B;FLOG = '0'B;
                     IF PERIOD = 0
                     THEN DO;
                              OP_STR(NR) = S_ANS;
                              FLAG = '1'B;
                              GO TO CON1;
                          END;
                     ELSE DO;
                              DIGET = SUBSTR(OPND,PERIOD+1);
                              FLOG = '1'B;
                              OP_STR(NR) = S_ANS_C(DIGET);
                              GO TO CON1;
                          END;
               CON1:NR_C = NR;
                     DO WHILE(SUBSTR(NR_C,1,1)=' ');
                        NR_C = SUBSTR(NR_C,2);
                     END;
                     OPND = NR_C;
                     NR = NR + 1;
                     IF NR > 25
                     THEN DISPLAY('OUT OF STORAGE');
```

```
                GO TO CONT;
        LAB(10):PERIOD = INDEX(OPND,'.');
                IF PERIOD = 0
                THEN IF FLAG
                     THEN OP_STR(NR)= M_ANS_C(DIGET);
                     ELSE OP_STR(NR)= M_ANS;
                ELSE DO;
                        DIGET = SUBSTR(OPND,PERIOD+1);
                        IF FLAG
                        THEN DO;
                               OP_STR(TOP)=S_ANS_C(DIGET);
                               OP_STR(NR) =M_ANS_C(DIGET);
                             END;
                        ELSE OP_STR(NR) = M_ANS_C(DIGET);
                   END;
                NR_C = NR;
                DO WHILE(SUBSTR(NR_C,1,1) = ' ');
                   NR_C = SUBSTR(NR_C,2);
                END;
                OPND = NR_C; NR = NR+1;
                IF NR>25 THEN DISPLAY('OUT OF STORAGE');
                GO TO CONT;
             END;
        ELSE DO;
                NR_C = NR;
                DO WHILE(SUBSTR(NR_C,1,1)= ' ');
                   NR_C = SUBSTR(NR_C,2);
                END;
                OP_STR(NR) = OPND;
                OPND = NR_C; NR = NR + 1;
                IF NR > 25 THEN DISPLAY('OUT OF STORAGE')
             END;
/*****************************************************************/
/*
      THE PRESENT VALUE OF TOP IS ADDED TO STACK AND THE
  VALUE OF OPND BECOMES TOP.
                                                                 */
/*****************************************************************/
      CONT: ALLOCATE STACK;
           IF TOP ¬= ''
           THEN STACK = SUBSTR(TOP,1,2);
           TOP = OPND;
           GO TO GET;
      END;
/*****************************************************************/
/*
      WHEN THE OPERATOR IS NOT A COMMA, THEN IT IS A
  RELATIONAL OPERATOR AND CONTROL IS PASSED TO THE SEGMENT
  OF CODE WHICH HANDLES THAT OPERATION.
                                                                 */
/*****************************************************************/
  ELSE GO TO LAB_L(INDEX(OPTRS,OPTR));
LAB_L(1):
  IF  STACK = '1 ' & TOP  ='1 '
  THEN FREE STACK;
  ELSE DO;
          TOP = '0 ';
          FREE STACK;
       END;
  GO TO GET;
LAB_L(2):
  IF  STACK = '1 ' | TOP = '1 '
  THEN TOP = '1 ';
  ELSE TOP = '0 ';
  FREE STACK;
  GO TO GET;
LAB_L(3):
  IF TOP = '1 '
  THEN TOP = '0 ';
  ELSE TOP = '1 ';
  GO TO GET;
LAB_L(4):
```

```
    IF OP_STR(STACK) > OP_STR(TOP)
    THEN TOP = '1 ';
    ELSE TOP = '0 ';
    FREE STACK;
    GO TO GET;
LAB_L(5):
    IF  OP_STR(STACK) = OP_STR(TOP)
    THEN TOP = '1 ';
    ELSE TOP = '0 ';
    FREE STACK;
    GO TO GET;
LAB_L(6):
    IF OP_STR(STACK) < OP_STR(TOP)
    THEN TOP = '1 ';
    ELSE TOP = '0 ';
    FREE STACK;
    GO TO GET;
LAB_L(8):
    IF  OP_STR(STACK) >= OP_STR(TOP)
    THEN TOP = '1 ';
    ELSE TOP = '0 ';
    FREE STACK;
    GO TO GET;
LAB_L(9):
    IF OP_STR(STACK) <= OP_STR(TOP)
    THEN TOP = '1 ';
    ELSE TOP = '0 ';
    FREE STACK;
    GO TO GET;
LAB_L(7): LAB_L(10):
    IF  MOD(OP_STR(STACK),OP_STR(TOP)) = 0
    THEN TOP = '1 ';
    ELSE TOP = '0 ';
    FREE STACK;
    GO TO GET;
LAB_L(11):
    OP_STR(TOP) = OP_STR(STACK) + OP_STR(TOP);
    FREE STACK;
    GO TO GET;
LAB_L(12):
    OP_STR(TOP) = OP_STR(STACK)- OP_STR(TOP);
    FREE STACK;
    GO TO GET;
LAB_L(13):
    OP_STR(TOP) = OP_STR(STACK) * OP_STR(TOP);
    FREE STACK;
    GO TO GET;
LAB_L(14):
    OP_STR(TOP) = OP_STR(STACK) / OP_STR(TOP);
    FREE STACK;
    GO TO GET;
LAB_L(0):
 /*****************************************************/
 /*
      WHEN COND_S IS NULL, THE INTERPRETATION IS COMPLETE
   AND TOP CONTAINS THE TRUTH VALUE OF THE STRING.
                                                      */
 /*****************************************************/
    IF TOP = '1 '
    THEN TRUTH = '1'B;
    ELSE TRUTH = '0'B;
END CON_INT;
```

BIBLIOGRAPHY

1. Brooks, F. P. Jr. and Oliver, P., "Evaluation of an Interactive Display System for Teaching Numerical Analysis," AFIPS Conference Proceedings, p. 525-535, 1969.

2. Bryan, G. L., "Computers and Education," Computers and Automation, v. 18, p. 16-19, March 1969.

3. Bryan, G. L., "Student-to-Student Interaction in Computer Time-Sharing Systems," Computers and Automation, v. 19, p. 18-23, March 1970.

4. Bushnell, D. D. and Allen, D. W., The Computer in American Education, John Wiley and Sons, Inc., 1967.

5. Coulson, J. E., Programmed Learning and Computer-Based Instruction, John Wiley and Sons, Inc., 1962.

6. Dunca, E. R., and others, Modern School Mathematics Structure and Use, California State Department of Education, 1970.

7. Entelek, Incorporated, Project Number 154-254, Computer-Assisted Instruction, A Survey of the Literature, by A. E. Hickey and J. M. Newton, January 1967.

8. Fenichel, R. R., Weizenbaum, J. and Yochelson, J. C., "A Program to Teach Programming," Communications of the ACM, v. 13, March 1970.

9. Ferguson, R. L., "Computer Assistance for Individualizing Instruction," Computers and Automation, v. 19, p. 27-29, March 1970.

10. Fry, E. B., Teaching Machines and Programmed Instruction, McGraw-Hill, 1963.

11. Gerald, R. W., Computers and Education, McGraw-Hill, 1967.

12. Harvard University Technical Report 6, Harvard Computer-Aided Instruction Laboratory, by L. M. Stolurow and T. I. Peterson, p. 8-13, March 1968.

13. Long, H. S. and Schwartz, H. A., "Instruction by Computer," Datamation, v. 12, p. 73-87, September 1966.

14. Macdonald, N., "The Role of Computers in Education," Computers and Automation, v. 13, p. 13-15, March 1964.

15. Philco-Ford Corporation Report PHO-TR307, Computer Assisted Instruction, by R. L. Balogh and D. L. Purdum, January 1968.

16. Rogers, J. L., "Current Problems in CAI," Datamation, v. 14, p. 28-33, September 1968.

17. Skinner, B. F., The Technology of Teaching, Appleton-Century-Crofts, 1968.

18. Spolsky, B., "Some Problems of Computer-Based Instruction," Behavioral Science, v. 11, p. 487-496, November 1966.

19. Suppes, P., "The Use of Computers in Education," Scientific American, v. 215, p. 207-220, September 1966.

20. System Development Corporation Report SP-933/001/00, Programmed Decisions in Programmed Instruction, by J. E. Coulson, 13 August 1962.

21. Systems Development Corporation Report SP-1653, Remote Computer Usage: Implications for Education, by T. C. Rowan, 12 January 1965.

22. System Development Corporation, Santa Monica, Calif. Report TM-2904/000/00, Using a Real-Time Operational Computer System as a Teaching Machine, by D. V. Springer, April 1966.

23. Training Research Laboratory University of Illinois Report TR 8, Essential Principles of Programmed Instruction, by L. M. Stolurow, p. 13, June 1965.

24. Training Research Laboratory Report 1, Teaching Machines and Computer-Based Systems, by L. M. Stolurow and D. Davis, May 1964.

25. Uhr, L., "Teaching Machine Programs that Generate Problems as a Function of Interaction with Students," 24th Conference Proceedings of the Association for Computing Machinery, p. 125-134, 1969.

26. Uhr, L., "The Compilation of Natural Language Text into Teaching Machine Programs," AFIPS Conference Proceedings, v. 26, p. 35-44, Fall 1964.

27. Uhr, L., "Toward the Compilation of Books into Teaching Machine Programs," IEEE Transactions on Human Factors, v. HFE-8, p. 81-84, June 1967.

28. Zinn, K. L., "Instructional Uses of Interactive Computer Systems," Datamation, v. 14, p. 22-27, September 1968.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3.	LTJG Robert Bolles, Code 53Bq Department of Mathematics Naval Postgraduate School Monterey, California 93940	1
4.	Assistant Professor George Heidorn, Code 55Hd Department of Operations Analysis Naval Postgraduate School Monterey, California 93940	1
5.	LT John C. Stewart, USN 185 Crestline Avenue Kalispell, Montana 59901	1

Unclassified

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1 ORIGINATING ACTIVITY (Corporate author)

Naval Postgraduate School
Monterey, California 93940

2a. REPORT SECURITY CLASSIFICATION

2b. GROUP

3 REPORT TITLE

RASCAL A Rudimentary Adaptive System for Computer-Aided Learning

4 DESCRIPTIVE NOTES (Type of report and inclusive dates)

Master's Thesis

5. AUTHOR(S) (First name, middle initial, last name)

John Christopher Stewart

6 REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
December 1970	152	28

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
b. PROJECT NO	
c.	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)
d.	

10 DISTRIBUTION STATEMENT

This document has been approved for public release and sale; its distribution is unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	Naval Postgraduate School Monterey, California 93940

13. ABSTRACT

The requirements of a Computer Aided Learning System which would be a reasonable assistant to the teacher are discussed. These ideas are implemented in a system entitled RASCAL, a Rudimentary Adaptive System for Computer Aided Learning. RASCAL replaces prepared frames used in previous systems with a description of questions to be asked and a tree of alternatives that might be helpful in assisting a student in answering a question. The actual questions are generated as a function of the system's interaction with the student, as is the selection of the branch to follow in aiding the student. The results obtained to date, while not extensive in their scope, indicate that a system such as RASCAL can be useful in the classroom.

DD FORM 1 NOV 65 1473 (PAGE 1)

S/N 0101-807-6811

Unclassified

Security Classification

A-31408

KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Computer-Assisted Instruction						
Computer-Aided Learning						
Learning						
Teaching						
Tutoring